FM-7 シリーズ

セブン

FM-7／NEW7／77

セブン　ニューセブン　セブン セブン

F-BASIC 解析マニュアル

フェーズIII サブシステム編

SHUWA
SYSTEM
TRADING
CO.,LTD.

注　意

FM-7 / NEW7 / 77

F-BASIC解析マニュアルフェーズIII

サブシステム編

SHUWA SYSTEM TRADING CO.,LTD

まえがき

富士通のＦＭ―７／８シリーズは，パーソナルコンピュータ業界において初めて本格的なデュアルＣＰＵシステムを採用して衆目を集めました．ＢＡＳＩＣその他の高級言語やＯＳを統轄して構文解釈・実行・入出力管理などを担当するメインシステムと，キーボード・画面処理・グラフィック等を専門に扱うサブシステムの双方に，８ビットＣＰＵの中では最上位機種とうたわれる６８０９ＣＰＵがそれぞれ搭載されています．こうすることによりタスクの分割がなされ，ハイコストパフォーマンスと，１６ビットマシン並みの高速性能が実現されました．

さて，２つのシステムのうち，メインシステムに関しては，ＦＭシリーズの人気上昇と相挨って様々な角度からの解説書が発行されてきました．これとは対照的にサブシステムはメインシステム以上に洗練されて魅力的なソフトウェアを身につけているにもかかわらず，その大部分は公開されていないのが現状です．サブシステムという存在自体がまったく目新しいことに加え，メインシステムの陰に隠れ容易にはアクセスできない点が主な原因です．

しかし，キー入力や様々な情報を受け取るための画面作成など，サブシステムとは私達ユーザにとって必要不可決な存在なのです．

本書は，コンピュータとして一つの生命を持ったサブシステムに初めての本格的なメスを入れ，その体系のすべてを記述しました．

一方，本書の第５章では，今まで盲点だったＦＭ―８から新シリーズＦＭ―ＮＥＷ７／７７に至るまでのメインシステム・サブシステム双方にわたる内部ルーチンの相違についても言及しました．互換性を最も重視している富士通では，新シリーズＦＭ―ＮＥＷ７／７７に対して，パッチによる修正を加えているだけです．殊に，サブシステムに関する限り１ビットたりとも変更は加えられていない点を強調しておきます．なお，紙面の都合により，ＦＭ－７／８のメインシステムの比較は続刊フェーズⅣ（応用編）に掲載させていただきます．

既刊フェーズⅠ（基礎編）／フェーズⅡ（探究編）と本書を併読することにより，サブシステムをはじめ，ＦＭマシン全体を自由自在に操ることができることでしょう．

読者の皆様においては，本書のより一層の御活用を切に望みます．

１９８４年7月

菊地　寿
南　泰浩
箕原　辰夫

参考文献

ＦＭ—７／ＮＥＷ７／７７　ユーザーズマニュアルシステム仕様　富士通㈱
ＦＭ７／８活用研究　Ｉ／Ｏ別冊　工学社
ＦＭ８活用研究　　　Ｉ／Ｏ別冊　工学社
ＦＭ—７／ＮＥＷ７／７７　マシン語入門マニュアル
ＦＭ—７　Ｆ—ＢＡＳＩＣ解析マニュアル　フェーズⅠ，Ⅱ
Ｎ88－ＢＡＳＩＣグラフィクス解析マニュアル
以上　秀和システムトレーディング㈱

目　　次

第１章　サブシステム転送命令の使い方……………………………………1
１—１　マシン語でサブシステムを直接アクセスする手順……………………3
１—２　ＴＥＳＴコマンドとサブシステム転送サブコマンド…………………10
　１—２—１　ＴＥＳＴコマンドについて…………………………………10
　１—２—２　サブシステム転送命令（ＭＯＶＥサブコマンド）……………12
　１—２—３　サブルーチンコール命令（ＪＳＲサブコマンド）……………16
　１—２—４　移動命令（ＪＭＰサブコマンド）……………………………23
　１—２—５　終了命令（ＥＮＤサブコマンド）……………………………26

第２章　サブシステムのアーキテクチャ…………………………………27
２—１　サブシステムの基本構造……………………………………………29
２—２　サブシステムのメモリマップ………………………………………30
２—３　メイン↔サブ間インタフェース……………………………………37
２—４　ＣＲＴＣによるＶＲＡＭ制御………………………………………49
２—５　高速スクロールの構造………………………………………………51
　２—５—１　表示されない部分とオフセットレジスタ……………………51
　２—５—２　スクロール…………………………………………………53
　２—５—３　スクロール活用法……………………………………………57
２—６　タイマ用ＮＭＩ（ノンマスカブルインタラプト）……………………60

第３章　サブシステムのモニタ……………………………………………61
３—１　サブシステムのソフトウェア体系……………………………………63
３—２　コマンド解釈ルーチンＣＭＤＣＴＬ…………………………………66
３—３　コンソール処理………………………………………………………73
　３—３—１　ＩＮＩＴ　＄０１　（＄Ｅ１４Ｅ）……………………………73
　３—３—２　ＥＲＡＳＥ　＄０２　（＄Ｅ２４Ｂ）…………………………86
　　　　　　ＥＲＡＳＥ２　＄０Ｄ　（＄Ｅ２４１）…………………………86
３—４　ＰＵＴ　ＧＥＴ…………………………………………………………103
　３—４—１　ＰＵＴ　＄０３　（＄Ｅ５Ｅ１）……………………………103

3—4—2　PUTのオーダサブルーチン……123
3—4—3　GET　$04　($E5E0)……133
3—4—4　GETのオーダサブルーチン……144
3—4—5　PUT，GET関連コマンドルーチン……158
3—4—6　Character Line　$20　($EE72)……161
3—4—7　PUT，GET汎用サブルーチン……162
3—5　グラフィック……183
3—5—1　LINE　$15　($EE98)……184
3—5—2　CHAIN　$16　($F0C4)……195
3—5—3　POINT　$17　($F412)……197
3—5—4　PAINT　$18　($FA8A)……199
3—5—5　グラフィック汎用サブルーチン……206
3—6　ブロックデータ処理……216
3—6—1　Graphic Cursor　$1F　($F67D)……217
3—6—2　SYMBOL　$19　($F103)……235
3—6—3　Get Block 1　$1B　($F286)……242
Get Block 2　$1D　($F3B6)……255
3—6—4　Put Block 1　$1C　($F446)……255
Put Block 2　$1E　($F589)……267
3—6—5　Get Character Block 1　$06　($ECF5)……267
Get Character Block 2　$08　($ECF4)……267
Put Character Block 1　$07　($ED72)……269
Put Character Block 2　$09　($ED71)……269
3—6—6　Change Color　$1A　($F21B)……273
3—6—7　ブロックデータ処理汎用サブルーチン……277
3—7　キーボード・タイマとその割込み処理……280
3—7—1　INKEY　$29　($FD5C)……280
3—7—2　Define String of PF　$2A　($FE02)……291
3—7—3　Get String of PF　$2B　($FE46)……294

３—７—４　Interrupt Control　＄２Ｃ　(＄ＦＥ６１)……………………295
３—７—５　Set Timer　＄３Ｄ　(＄ＦＥ７Ｃ)…………………………301
Read Timer　＄３Ｅ　(＄ＦＥＡ４)…………………………301

第４章　アプリケーションプログラム……………………………………309
４—１　サブシステムへのプログラム転送……………………………………311
４—２　サブシステムでプログラムを動かすときの注意…………………………313
４—３　サブシステムコールプログラム………………………………………316
４—４　画面シフト………………………………………………………………330
４—５　画面スクロール…………………………………………………………336
４—６　画面拡張…………………………………………………………………341

第５章　ＦＭ−７／ＮＥＷ７／７７とＦＭ−８との比較…………………347
５—１　ＦＭ—７とＦＭ—８のサブシステムの比較……………………………349
５—１—１　ＦＭ—７とＦＭ—８のサブシステムの比較…………………………349
５—１—２　サブシステムのワークエリア，アドレス対照表……………………351
５—２　ＦＭ—７／ＮＥＷ７／７７内部ルーチンの比較…………………………365

第1章

サブシステム転送命令の使い方

サブシステム操作のイロハから――――

サブシステムの扉を叩き，サブシステムを自由自在に操るためには，転送命令をはじめとするＴＥＳＴコマンドを熟知し，使いこなせるようになることが必要条件です．
この第１章では，サブシステム操作のイロハから，ＴＥＳＴコマンドの使用法に至るまでについて述べていきます．

PHASE Ⅲ
SUBSYSTEM

１－１　マシン語でサブシステムを直接アクセスする手順

ＦＭ－７のサブシステムは，普通，メインシステムとは切り離されており，サブシステム内のデータ・ステータス等はメイン側から直接アクセスすることはできません．所定の手続きに従い，共有ＲＡＭを通すことによって，はじめて，メイン側とサブ側との情報交換が可能となるのです．通常は，ＢＩＯＳ上に設けられたＳＵＢＩＮ・ＳＵＢＯＵＴといったコマンドを用いることによって，ＢＩＯＳ上の標準手続きがサブシステムとの直接の情報交換を肩代りしてくれるので，ユーザは，メイン↔サブ間インタフェース回路の信号線を意識することなく，サブシステムとのデータのやりとりが可能です．しかしながら，ＢＩＯＳのＳＵＢＩＮ・ＳＵＢＯＵＴでは小回りが効かず，実行速度も若干落ちてしまいます．

サブシステムとの情報交換の手続きは，たいして面倒なものではないので，是非とも修得していただきたいものです．

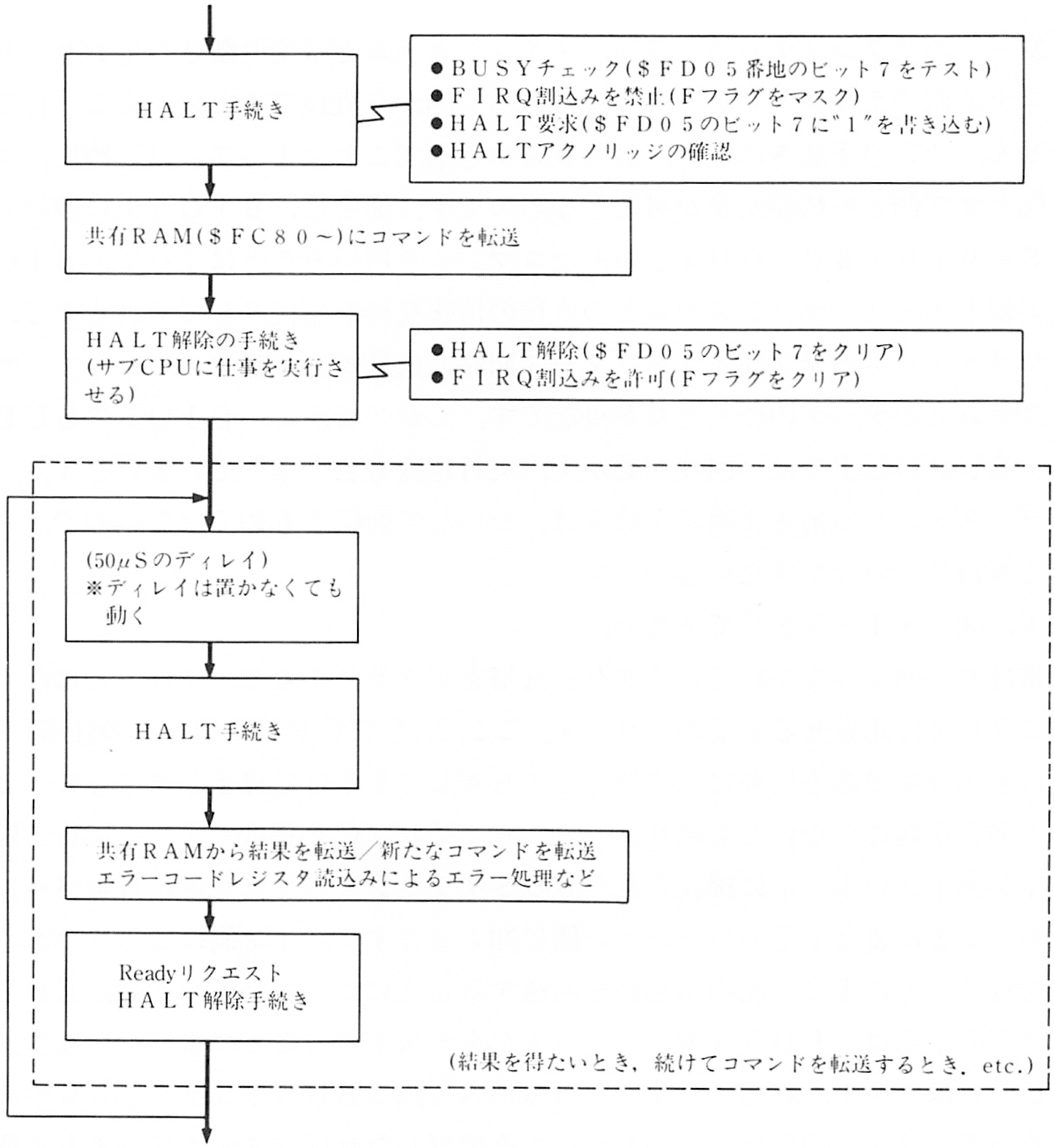
まず，図１・１・１を見て下さい．

通常はサブ側につながれている共有ＲＡＭをアクセスするためには，とにかくサブＣＰＵを停止させる必要があります．ここで，サブＣＰＵが何らかの仕事（タスク）を実行中である場合に，突如としてＨＡＬＴをかけて停止させてしまっては，不都合が起こらないとも限りません．そこでサブＣＰＵは，タスク実行の際にメイン側Ｉ／Ｏポートに接続されたＢＵＳＹフラグという信号線をＨレベルにしておくことによってその旨をメイン側に知らせます．メイン側はこのフラグの確認を行うことにより，先の不都合を回避するようになっているのです．ＢＵＳＹフラグの確認は＄ＦＤ０５番地のビット７をテストすることにより行います．サブＣＰＵはタスクを終えてコマンド待ちに入る際にＢＵＳＹフラグをＬレベルにするので，メイン側では，このＬレベルを検出したサブシステムにＨＡＬＴ要求（リクエスト）を行うという，アルゴリズムを持った手続きを用意しなければなりません．ＨＡＬＴ要求は，＄ＦＤ０５番地のビット７に“１”を書き込み，Ｈレベルにすることによって行います．

さて，サブＣＰＵがＨＡＬＴを受け入れると，ＢＵＳＹフラグは自動的にＨレベルに戻ります．これはハードウェアによるものです．すなわち，ＢＵＳＹフラグは，ＨＡＬＴアクノリッジ状態であることをメイン側に知らせる役目を兼ねており，ＨＡＬＴ要求後のアクノリッジの確認は，やはり＄ＦＤ０５のビット７をテストすることによって行います．

サブＣＰＵ停止後，いよいよ共有ＲＡＭを通じて情報交換が可能となります．この際に注意しなければならないのは，共有ＲＡＭのアドレスです．メイン側か

図1・1・1　サブシステムをアクセスする手順



ら見た共有ＲＡＭの位置（ロケーション）は，＄ＦＣ８０〜＄ＦＣＦＦ番地となっているのに対し，サブ側から見た共有ＲＡＭは，＄Ｄ３８０〜＄Ｄ３ＦＦ番地となります．この違いは，かなりの上級者でも，思わぬ混乱をひき起こすことがあり，厄介です．さて，最初のアクセスでは，共有ＲＡＭ上に，サブシステムに対するコマンド・パラメータ等（場合によっては，ユーザ自身が作ったプログラムやデータなどを送出することもある）を転送し，その後，ＨＡＬＴを解除することによってコマンド等をサブＣＰＵに実行させます．さらに結果を得たい時や続けてデータを転送する場合には，２回，３回…とＨＡＬＴをかけ直します．サブＣＰＵに続けてＨＡＬＴをかける時には，間にディレイ（遅延）を置く必要が

あるなどといわれますが，このディレイは全く不要です．図1・1・1および例1・1・1にもディレイがはさんでありますが，これは，例1・1・2のＢＩＯＳ内部ルーチンを使ったサンプルとの比較対照のために残しておいたもので，実際にプログラムを打ち込む時には省いてかまいません．

このディレイのほかに気をつけなければならないのは，結果の読込みの際に，＄ＦＣ８０（サブ側の＄Ｄ３８０）番地のビット０～６までの７ビットが，タスク実行時のエラーを知らせるレジスタとなっていること，またＨＡＬＴ後に新たなコマンドを送出しない場合には，＄ＦＣ８０番地のビット７を立てておく（この操作を Ready リクエスト といいます）必要があることなどです．一方，サブシステムに送出するコマンド・パラメータ群は＄ＦＣ８２（サブ側の＄Ｄ３８２）番地以下にセットして，＄ＦＣ８１（サブ側の＄Ｄ３８１）はあけておきます．このアドレスは，割込み処理のステータスレジスタとして利用されていますが，これについては折をみてふれていくこととします．

それでは，実際のプログラム例で見ることにしましょう．

例1・1・1　サブシステムＣＣＯＬＯＲ（Change Color）コマンドを使用した簡単なサンプルです．このコマンドに対するＢＡＳＩＣ命令はサポートされてはいないので，目新しく感じるかもしれません．実行速度が遅いのでゲームなどには不向きですが，試作の段階での色調整など画面上で枠を指定し，各種の色の塗換えなどに使用できます．以下の例では「白→紫，黒→黄」に塗り換えますが，＄４０８Ｂ番地以下のパラメータを書き換えることによって様々な応用を試みて下さい．

```
                                   NAM     CCOLOR  *Sample Program 1.1.1
                                   OPT     MEM,NOGEN * Change COLOR
4000                               ORG     $4000   *This Program is
                         *                         *    POSITION-INDEPENDANT
                         *
           FD05          BSYHLT EQU    $FD05   *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
           FC80          SH.RAM EQU    $FC80   *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
           9BDB          OUTSTR EQU    $9BDB   *ﾓｼﾞﾚﾂ ﾉ ｼｭﾂﾘｮｸ
                         *                         *  FM-8..$9FDF
           AC3D          OUT_A  EQU    $AC3D   *A reg ｦ 16ｼﾝｽｳ ﾃﾞ ｼｭﾂﾘｮｸ
                         *                         *  FM-8..$AFE9
                         *
                         *
           4000          CCOLOR EQU    *       *Start of MAIN Program
4000 30    8D 0084                 LEAX    CCDATA,PCR * ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> X reg
4004 108E  FC80                    LDY     #SH.RAM * ｷｮｳﾕｳRAMｾﾝﾄｳ --> Y reg
4008 E6    8D 007B                 LDB     LENGTH,PCR * ﾃﾝｿｳﾊﾞｲﾄｽｳ   --> B reg
400C 8D    1D    402B              BSR     CM_TFR  * ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
400E 8D    3C    404C              BSR     HRST    * HALT ﾉ ｶｲｼﾞｮ
4010 8D    23    4035              BSR     HLTWDL  * HALT with Delay
4012 8D    4C    4060              BSR     RDSTAT  * STATUS ﾉ ﾖﾐｺﾐ
4014 26    05    401B              BNE     ERROR   * ｴﾗｰ ﾅﾗ ERROR ﾍ...
                         *
4016 8D    3F    4057   CC.END BSR     REDYRQ  * ｻﾌﾞｼｽﾃﾑ ﾆ Ready Request
4018 8D    32    404C              BSR     HRST    * HALT ｶｲｼﾞｮ
401A 39                            RTS             *End of MAIN Program
```

```
10270                           *
10280  401B 34   02             ERROR   PSHS   A          *Error ﾉ ｼｮﾘ
10290  401D 8D   F7    4016             BSR    CC.END     * Ready Request
10300  401F 30   8C 43                  LEAX   <ERRMSG-1,PCR ｴﾗｰﾒｯｾｰｼﾞｾﾝﾄｳ --> X reg
10310  4022 BD   9BDB                   JSR    OUTSTR     *     ﾒｯｾｰｼﾞ ﾉ ｼｭﾂﾘｮｸ
10320  4025 35   02                     PULS   A
10330  4027 BD   AC3D                   JSR    OUT_A      * ERROR Number ｼｭﾂﾘｮｸ
10340  402A 39                          RTS
10350                           *
10360  402B 8D   0D    403A     CM_TFR  BSR    HLT        *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
10370  402D A6  80                      LDA    ,X+
10380  402F A7  A0                      STA    ,Y+
10390  4031 5A                          DECB
10400  4032 26   F9    402D             BNE    *-5
10410  4034 39                          RTS
10420                           *
10430  4035 86   18             HLTWDL  LDA    #24        *HALT With Delay
10440  4037 4A                          DECA              * (50 ﾏｲｸﾛｾｶﾝﾄﾞ ﾉ ﾃﾞｨﾚｲ)
10450  4038 26   FD    4037             BNE    *-1
10460  403A B6   FD05           HLT     LDA    $FD05      *HALT Request to ｻﾌﾞｼｽﾃﾑ
10470  403D 2B   FB    403A             BMI    HLT        * if BUSY then Wait
10480  403F 1A   50                     ORCC   #$50       * FIRQ,IRQ ｷﾝｼ
10490  4041 86   80                     LDA    #$80       * BUSYﾌﾗｸﾞ ﾉ ｾｯﾄ
10500  4043 B7   FD05                   STA    $FD05
10510  4046 B6   FD05                   LDA    $FD05      * Wait untill HALT ｱｸﾉﾘｯｼﾞ
10520  4049 2A   FB    4046             BPL    *-3
10530  404B 39                          RTS
10540                           *
10550  404C 34   02             HRST    PSHS   A          *HALT ﾉ ｶｲｼﾞｮ
10560  404E 4F                          CLRA
10570  404F B7   FD05                   STA    BSYHLT
10580  4052 1C   AF                     ANDCC  #$AF       * FIRQ,IRQ ｷｮｶ
10590  4054 35   02                     PULS   A
10600  4056 39                          RTS
10610                           *
10620  4057 F6   FC80           REDYRQ  LDB    SH.RAM     *READY Request to ｻﾌﾞｼｽﾃﾑ
10630  405A CA   80                     ORB    #$80
10640  405C F7   FC80                   STB    SH.RAM
10650  405F 39                          RTS
10660                           *
10670  4060 B6   FC80           RDSTAT  LDA    $FC80      *Read STATUS
10680  4063 84   7F                     ANDA   #$7F
10690  4065 39                          RTS
10700                           *
10710  4066      0D0A           ERRMSG  FDB    $0D0A      *Error Message
10720  4068      2A                     FCC    '*** Error Occured ! ***  No.=$'
10730  4086      00                     FCB    0
10740                           *
10750  4087      10             LENGTH  FCB    ..DATA-CCDATA *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
10760  4088      0000           CCDATA  FDB    $0000      *ﾃﾝｿｳﾃﾞｰﾀ
10770  408A      1A             .CMDCD  FCB    $1A        * CCOLOR ｺﾏﾝﾄﾞｺｰﾄﾞ
10780  408B      00A0           .STRTX  FDB    160        * ｶｲｼｲﾁ ﾉ X ｻﾞﾋｮｳ
10790  408D      0032           .STRTY  FDB    50         * ｶｲｼｲﾁ ﾉ Y ｻﾞﾋｮｳ
10800  408F      01DF           ..ENDX  FDB    479        * ｼｭｳﾘｮｳｲﾁ ﾉ X ｻﾞﾋｮｳ
10810  4091      0095           ..ENDY  FDB    149        * ｼｭｳﾘｮｳｲﾁ ﾉ Y ｻﾞﾋｮｳ
10820  4093      02             ..NOCL  FCB    2          * ﾇﾘｶｴ ｶﾗｰ ﾉ ｶｽﾞ
10830  4094      07             ..CL01  FCB    7          * Old Color 1 (WHITE)
10840  4095      03             ..CLN1  FCB    3          * New Color 1 (VIOLET)
10850  4096      00             ..CL02  FCB    0          * Old Color 2 (BLACK)
10860  4097      06             ..CLN2  FCB    6          * New Color 2 (YELLOW)
10990            4098           ..DATA  EQU    *          *End of DATA
11000            4000                   END    CCOLOR
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=4097
PROGRAM ENTRY ADDR=4000
```

以上の例1・1・1において，リスト上のラベル名と，図1・1・1上の各処理ステップとは次表のように対応しております．

ラベル名	図・1・1・1上のステップ	BIOSエントリ※
HLT	HALT手続	$F899
HLTWDL	ディレイ＋HALT手続き	$F894
HRST	HALT解除手続き	$F8B0
REDYRQ	Ready リクエスト	$F8BF
RDSTAT	エラーコードレジスタ読込	$F8C8
CM_TFR	HALT手続き＋共有RAM上にコマンド等を転送	

※メイン側ROM上のBIOS内に設けられている，等価なサブルーチンのエントリアドレス．以下に述べる本文参照．

さて，ここに列挙した標準サブルーチン群は，BIOS内部にも既に用意されていて，それらをうまく使えば，例1・1・2のようにスッキリとしたプログラムを書くことができます（各サブルーチンのエントリアドレスは上表の最右欄に掲げたとおりです）．ただし，あらかじめDPレジスタに＄FDを代入しておかなければならない点だけ，特に注意してください．ラベルの付け方は例1・1・1と全く同様です．また，参考のため，BIOS内の各サブルーチンのソースリストを掲載しておきます．

例1・1・2　BIOSを利用した例

```
                          NAM    CCOLOR  *Sample Program 1.1.2
                          OPT    MEM,NOGEN * Change COLOR
4000                      ORG    $4000   *This Program is
                 *                       *    POSITION-INDEPENDANT
                 *
                 *  BIOS ﾅｲﾌﾞﾙｰﾁﾝ
                 *
           F894  HLTWDL EQU      $F894   *HALT With Delay
           F899  HLT    EQU      $F899   *HALT
           F8B0  HRST   EQU      $F8B0   *Halt RESET
           F8BF  REDYRQ EQU      $F8BF   *Ready Request to ｻﾌﾞｼｽﾃﾑ
           F8C8  RDSTAT EQU      $F8C8   *Read Status of ERROR-reg.
                 *
                 *  ｿﾉﾀ ﾉ ﾛｹｰｼｮﾝ
                 *
           FD05  BSYHLT EQU      $FD05   *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
           FC80  SH.RAM EQU      $FC80   *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
           9BDB  OUTSTR EQU      $9BDB   *ﾓｼﾞﾚﾂ ﾉ ｼｭﾂﾘｮｸ
           AC3D  OUT_A  EQU      $AC3D   *A reg ｦ 16ｼﾝｽｳ ﾃﾞ ｼｭﾂﾘｮｸ
                 *
                 *  PROGRAM
                 *
           4000  CCOLOR EQU      *       *Start of MAIN Program
4000 86    FD           LDA      #$FD    *DP reg ﾆ $FD( I/O ﾘｮｳｲｷ ) ｦ
4002 1F    8B           TFR      A,DP    * ｶｸﾉｳｽﾙ ( BIOS ﾖｳ ) !!!!!
4004 30    8D 005C      LEAX     CCDATA,PCR * ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> X reg
4008 108E  FC80         LDY      #SH.RAM * ｷｮｳﾕｳRAMｾﾝﾄｳ --> Y reg
400C E6    8D 0053      LDB      LENGTH,PCR * ﾃﾝｿｳﾊﾞｲﾄｽｳ   --> B reg
4010 8D    25   4037    BSR      CM_TFR  * ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
4012 BD    F8B0         JSR      HRST    * HALT ﾉ ｶｲｼﾞｮ
4015 BD    F894         JSR      HLTWDL  * HALT with Delay
4018 BD    F8C8         JSR      RDSTAT  * STATUS ﾉ ﾖﾐｺﾐ
401B 26    0A   4027    BNE      ERROR   * ｴﾗｰ ﾅﾗ ERROR ﾍ...
                 *
401D BD    F8BF  CC.END JSR      REDYRQ  * ｻﾌﾞｼｽﾃﾑ ﾆ Ready Request
```

```
4020 BD   F8B0            JSR    HRST      * HALT ｶｲｼﾞｮ
4023 4F                   CLRA             * DP reg ｦ
4024 1F   8B              TFR    A,DP      *   ﾓﾄ ﾆ ﾓﾄﾞｽ !!!!!
4026 39                   RTS              *End of MAIN Program
                   *
4027 34   02       ERROR  PSHS   A         *Error ﾉ ｼｮﾘ
4029 8D   F2   401D       BSR    CC.END    * Ready Request
402B 30   8C 13           LEAX   <ERRMSG-1,PCR ｴﾗｰﾒｯｾｰｼﾞｾﾝﾄｳ --> X reg
402E BD   9BDB            JSR    OUTSTR    *     ﾒｯｾｰｼﾞ ﾉ ｼｭﾂﾘｮｸ
4031 35   02              PULS   A
4033 BD   AC3D            JSR    OUT_A     * ERROR Number ｼｭﾂﾘｮｸ
4036 39                   RTS
                   *
4037 BD   F899     CM_TFR JSR    HLT       *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
403A A6   80              LDA    ,X+
403C A7   A0              STA    ,Y+
403E 5A                   DECB
403F 26   F9   403A       BNE    *-5
4041 39                   RTS
                   *
                   *  Error Message
                   *
4042      0D0A     ERRMSG FDB    $0D0A     *Error Message
4044      2A              FCC    '*** Error Occured ! ***  No.=$'
4062      00              FCB    0
                   *
                   *  RCB Block
                   *
4063      10       LENGTH FCB    ..DATA-CCDATA *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
4064      0000     CCDATA FDB    $0000     *ﾃﾝｿｳﾃﾞｰﾀ
4066      1A       .CMDCD FCB    $1A       * CCOLOR ｺﾏﾝﾄﾞｺｰﾄﾞ
4067      00A0     .STRTX FDB    160       * ｶｲｼｲﾁ ﾉ X ｻﾞﾋｮｳ
4069      0032     .STRTY FDB    50        * ｶｲｼｲﾁ ﾉ Y ｻﾞﾋｮｳ
406B      01DF     ..ENDX FDB    479       * ｼｭｳﾘｮｳｲﾁ ﾉ X ｻﾞﾋｮｳ
406D      0095     ..ENDY FDB    149       * ｼｭｳﾘｮｳｲﾁ ﾉ Y ｻﾞﾋｮｳ
406F      02       ..NOCL FCB    2         * ﾇﾘｶｴ ｶﾗｰ ﾉ ｶｽﾞ
4070      07       ..CLO1 FCB    7         * Old Color 1 (WHITE)
4071      03       ..CLN1 FCB    3         * New Color 1 (VIOLET)
4072      00       ..CLO2 FCB    0         * Old Color 2 (BLACK)
4073      06       ..CLN2 FCB    6         * New Color 2 (YELLOW)
          4074     ..DATA EQU    *         *End of DATA
          4000            END    CCOLOR
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=4073
PROGRAM ENTRY ADDR=4000
```

参　考　ＢＩＯＳ内の，関係ルーチンのソースリスト

```
                                 NAM    BIOS.S    *ｻﾝｺｳﾘｽﾄ 1.1.2
                                 OPT    NOMEM,OBJ
F894                             ORG    $F894     *BIOSﾅｲﾌﾞｻﾌﾞﾙｰﾁﾝ
                          *
           00FD                  SETDP  $FD       *BIOS ﾅｲﾌﾞ ﾉ DP-reg ﾉ ｱﾀｲ !
                          *
           FD05           BSYHLT EQU    $FD05     *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
           FC80           SH.RAM EQU    $FC80     *ｷｮｳﾕｳ RAM ｾﾝﾄｳ
                          *
           F894           HLTWDL EQU    *         *HALT with DELAY
F894 86    18                    LDA    #$18      *  ( 50 ﾏｲｸﾛｾｶﾝﾄﾞ ﾉ DELAY )
F896 4A                   HLTWD1 DECA
F897 26    FD    F896            BNE    HLTWD1
           F899           HLT    EQU    *         *HALT Request to ｻﾌﾞｼｽﾃﾑ
F899 96    05                    LDA    BSYHLT
F89B 2B    FC    F899            BMI    HLT
F89D 1A    40                    ORCC   #$40      *FIRQ ｷﾝｼ
F89F 86    80                    LDA    #$80
F8A1 97    05                    STA    BSYHLT
F8A3 34    04                    PSHS   B
F8A5 C6    08                    LDB    #$08      *ﾌﾟﾛｸﾞﾗﾑ ﾉ ｼﾝｶ ﾉ ｱﾄ !!!
F8A7 5A                   ..HLT1 DECB
F8A8 27    04    F8AE            BEQ    HLTRTS
F8AA 96    05                    LDA    BSYHLT
F8AC 2A    F9    F8A7            BPL    ..HLT1
                          *
```

```
10270  F8AE 35    84          HLTRTS PULS    PC,B      *ﾘﾀｰﾝ
10280                         *
10290             F8B0        HRST   EQU     *         *HALT RESET
10300  F8B0 34    02                 PSHS    A
10310  F8B2 86    00                 LDA     #$00
10320  F8B4 97    05                 STA     BSYHLT
10330  F8B6 1C    BF                 ANDCC   #$BF      *FIRQ ｷｮｶ
10340  F8B8 35    02                 PULS    A
10350  F8BA 8D    02    F8BE         BSR     HRST1     *Delay ( ｼﾝｶ ﾉ ｱﾄ!!! )
10360  F8BC 8D    00    F8BE         BSR     HRST1
10370             F8BE        HRST1  EQU     *
10380  F8BE 39                       RTS
10390                         *
10400             F8BF        REDYRQ EQU     *         *Ready Request
10410  F8BF F6    FC80               LDB     SH.RAM
10420  F8C2 CA    80                 ORB     #$80
10430  F8C4 F7    FC80               STB     SH.RAM
10440  F8C7 39                       RTS
10450                         *
10460             F8C8        RDSTAT EQU     *         *Read Status of ERROR-reg.
10470  F8C8 B6    FC80               LDA     SH.RAM
10480  F8CB 84    7F                 ANDA    #$7F
10490  F8CD 39                       RTS
10500                         *
10510                                END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=F894
PROGRAM END   ADDR=F8CD
PROGRAM ENTRY ADDR=****
```

先程もふれましたが，上掲リスト中のディレイは，現在のＦＭ－７ではすべて不要です．なぜこんな余計なものがあるかというと，ＦＭ－７がまだ試作品段階の頃にはディレイを入れないと動かないケースがあったからなのです．現在はハードウェアが完全なものとなってディレイを入れなくてもちゃんと動くようになったのですが，ＢＩＯＳのＲＯＭ上に痕跡として残ってしまったものです．同様にＨＬＴのアクノリッジテストの際，７回待ってもアクノリッジが立たない場合は，強制的にリターンしてしまう構造になっているというのも，試作品段階でハードウェアのトラブルにより無限ループに陥るのを防ぐ目的があったため，と推察されます．

プログラム中にマシンの進化の跡が見受けられるのは，ちょうど人間に尻尾の痕跡があるのと同様です．

1－2　ＴＥＳＴコマンドとサブシステム転送サブコマンド

1－2－1　ＴＥＳＴコマンドについて

もう御存知の方も多いかと思いますが，ＦＭ－７のサブシステムには，システム開発時およびメンテナンス時のために設置されたと思われるＴＥＳＴコマンドというものがあり，他のコマンド群と同様の手順で使用することができます（ＦＭ－７のマニュアル類にはＴＥＳＴコマンドの存在すら書かれておりません）．

ＴＥＳＴコマンドのコマンドコードは＄３Ｆで，このコマンドの下に，転送命令(ＭＯＶＥ)，サブルーチンコール(ＪＳＲ)，移動命令(ＪＭＰ)，終了命令(ＥＮＤ)の４つのサブコマンドが用意されております．ユーザは１つのＴＥＳＴコマンドの支配下で，この４つのサブコマンドを自在に組み合わせて，繰り返し使用することができます．ただし当然のことながら，終了命令(END サブコマンド)の後に他のサブコマンドを置いても無意味です．一般によく話題になっているサブシステム転送命令というのは，実はＴＥＳＴコマンド下のＭＯＶＥサブコマンドのことに他ならないのです．

ＴＥＳＴコマンドの一般的なフォーマットを図１・２・１に示します．

なお，ＦＭ－８では，ＴＥＳＴコマンドを起動させるためには，キーワード文字列「ＹＡＭＡＵＣＨＩ」を＄Ｄ３８３～Ｄ３８Ａ番地（メイン＄ＦＣ８３～）に入力しておく必要がありましたが，ＦＭ－７では任意の文字列で起動させることができます．ただし８文字分の空間をあけておく必要があり，＄Ｄ３８Ｂ番地（メイン＄ＦＣ８Ｂ)から，サブコマンドを書き始めなければならないのはＦＭ－８の場合と同様です．

図1・2・1　TESTコマンドおよびそのサブコマンドのフォーマット

① TESTコマンドのフォーマット

共有RAMアドレス	
$D380	エラーコードレジスタ Ready Request (Don't Care)
$D381	ステータスレジスタ CONTフラグ (Don't Care)
$D382	TESTコマンドコード ($3F)
$D383 〜 $D38A	パスワード FM−8……「YAMAUCHI」 FM−7……任意の8バイトの文字列
$D38B ：	サブコマンド1(ENDを除く)
：	サブコマンド2(ENDを除く)
：	：
：	サブコマンドn
：	ENDサブコマンド(終了命令)

② 転送命令(MOVEサブコマンド)のフォーマット

アドレス相対値	
0	コード$91
+1 +2	転送元アドレス
+3 +4	転送先アドレス
+5 +6	転送バイト数

③ 移動命令(JMPサブコマンド)のフォーマット

アドレス相対値	
0	コード$92
+1 +2	次のサブコマンドがある先頭アドレスをさすポインタ

④ サブルーチンコール命令(JSRサブコマンド)のフォーマット

0	コード$93
+1 +2	マシン語サブルーチンのエントリアドレス

⑤ 終了命令(ENDサブコマンド)のフォーマット

0	コード$90

Ｉ－２－２　サブシステム転送命令（ＭＯＶＥサブコマンド）

この命令は，サブシステムのメモリ間のブロック転送を行うものですが，転送可能なメモリは制限を受けます．すなわち，ＶＲＡＭエリアのメモリは通常，転送元としても転送先としても指定することはできません．ＶＲＡＭエリアのメモリをアクセスするためには，ＶＲＡＭアクセスフラグというものを立てておく必要があるわけですが，ＴＥＳＴコマンドの制御下に入る際，このフラグはオフになっています．次に述べるＪＳＲコマンド等で，あらかじめＶＲＡＭアクセスフラグを立てておかない限り，ＶＲＡＭエリアの転送を行ってもメモリの正常なリード／ライト動作が行われずうまくいきません．一方，Ｉ／Ｏ領域の読書きは特殊な意味を持っていますので，この領域も避ける必要があります．

従って，通常このコマンドは，共有ＲＡＭ上のユーザプログラムやデータなどをサブシステム上の別のＲＡＭ上にロードしておいて，その後で，ＪＳＲ命令により呼び出すための準備をしたり，サブシステム内のフラグ類やバッファデータ等の情報を共有ＲＡＭ上に送って，メイン側で参照する手段にするなどの用途に使われることになります．

転送命令の使い方に慣れるためのサンプルとして，例１・２・１を掲載します．転送命令は，次の節で述べるサブルーチンコール命令（ＪＳＲサブコマンド）と合わせて修得すると，サブシステムを自由自在に使いこなすための強力な武器となります．そのためには，自分のアイデアをどんどんプログラムにして，実際に走らせてみることです．初めのうちは何回も失敗するでしょうが，サブシステムの場合，失敗して暴走したとしても，きれいな色とりどりの模様が画面に現われ，リズミカルな!?ＢＥＥＰ音が発生することもありますので，失敗も苦になりません．

例１・２・１　このサンプルは転送命令（ＭＯＶＥサブコマンド）を用いて，サブシステムモニタ（サブシステムの制御プログラム：＄Ｅ０００～＄ＦＦＦＦ番地のＲＯＭ上に書かれている）をそっくりそのままメイン側メモリに移してしまうサブルーチンプログラムです．サブシステムの制御プログラムのソースリストは，既に公開されていますが，独自のオリジナルソースリストを作りたいとか，サブシステムのＲＯＭ上で，あるパターンを検索したいなどと思っている方は，このサンプルプログラムをおおいに利用して下さい．また，プログラム先頭の部分を少し変えるだけで，ＶＲＡＭおよびＩ／Ｏ領域を除いたサブ側メモリを，任意のアドレスから任意のバイト数だけメイン側に読み込む動作を行うサブルーチ

ンパッケージとすることが可能です. データは６４バイト単位で転送されますが, ６４バイト未満の端数があっても, 最初にその端数分だけ転送するようにうまく作られておりますので, 安心して下さい.

このプログラム自身は, ポジションインディペンデントに作られておりますが, 大容量の転送を行うときなどは, スタックを破壊したりしないように, バッファのとり方には充分に注意を払って下さい. 特に, このプログラムをそのまま使用する場合には, 必ず, 「ＣＬＥＡＲ,＆Ｈ３ＦＦＦ」などとしてスタックエリアをずらしてから走らせて下さい. プログラム中には, スタックをワークエリアとして使用するといった, やや高級な技法も使用されていますので, リエントラントなプログラムを書く勉強をなさっている方などは, 参考にしていただきたいと思います. 特に, ＳレジスタとＵレジスタの使い分けや, ラベルの付け方などに注目して下さい.

なお, このプログラムは, ＦＭ－８にも対応できるように作られており, ＦＭ－８で使用する場合には, メインＲＯＭ内部ルーチンのエントリアドレスの＄９ＢＤＢ, ＄ＡＣ３Ｄを, それぞれ＄９ＦＤＦ, ＄ＡＦＥ９に変更するだけでよいのです.

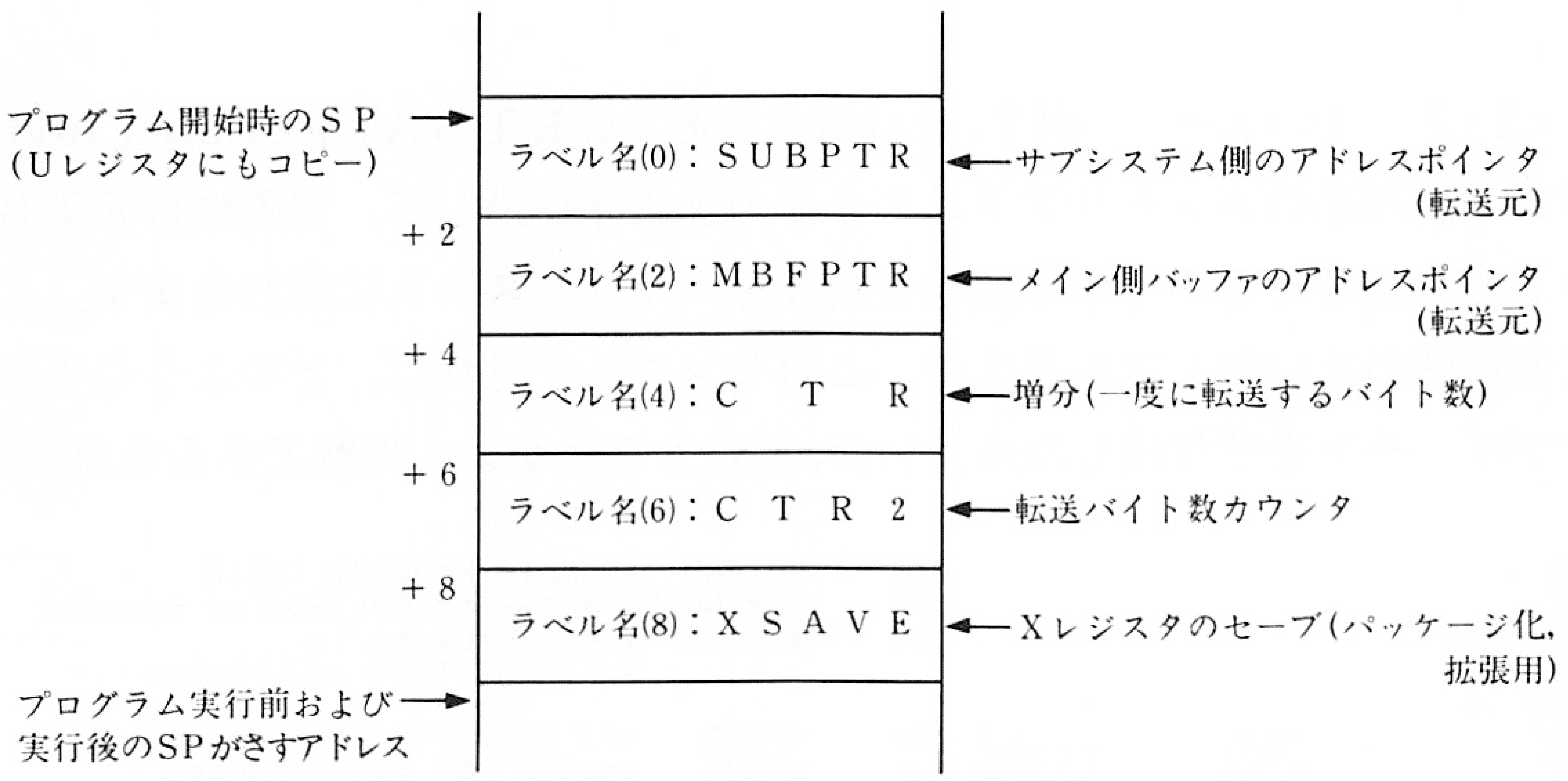
```
                              NAM     SM_TFR  *Sample Program 1.2.1
                              OPT     MEM,NOGEN SUB to MAIN Transfer
4000                          ORG     $4000   *This Program is
                      *                       *     POSITION-INDEPENDANT
                      *
                      *  ｽﾀｯｸｼﾞｮｳ ﾉ ﾜｰｸｴﾘｱ ﾉ ﾃｲｷﾞ
                      *
           0000       SUBPTR EQU      $0000   *SUB ｶﾞﾜ ﾎﾟｲﾝﾀ
           0002       MBFPTR EQU      $0002   *MAINｶﾞﾜ ﾎﾟｲﾝﾀ
           0004       CTR    EQU      $0004   *ｿﾞｳﾌﾞﾝ
           0006       CTR2   EQU      $0006   *DATA ｶｳﾝﾀ
           0008       XSAVE  EQU      $0008   *Xreg ｾｰﾌﾞ ﾖｳ
                      *
                      *  ﾒｲﾝ･ｻﾌﾞ ｲﾝﾀﾌｪｰｽ
                      *
           FD05       BSYHLT EQU      $FD05   *ﾒｲﾝ･ｻﾌﾞ ｲﾝﾀﾌｪｰｽ
           FC80       SH.RAM EQU      $FC80   *ｷｮｳﾕｳRAM ｾﾝﾄｳｱﾄﾞﾚｽ
           FCC0       SH.BUF EQU      $FCC0   *ｷｮｳﾕｳRAM ﾃﾝｿｳﾊﾞｯﾌｧ
                      *
                      *  ﾒｲﾝROM ﾅｲﾌﾞﾙｰﾁﾝ
                      *
           9BDB       PO     EQU      $9BDB   *ﾓｼﾚﾂﾋｮｳｼﾞ
           AC3D       OUT_A  EQU      $AC3D   *Areg ﾋｮｳｼﾞ
                      *
                      *  ﾌﾟﾛｸﾞﾗﾑ
                      *
           4000       SM_TFR EQU      *       *ﾒｲﾝ ﾙｰﾁﾝ
4000 34    10                PSHS     X       *Xreg ｾｰﾌﾞ
4002 CC    2800              LDD      #$2800  *ﾃﾝｿｳﾊﾞｲﾄｽｳ ( 10K )
4005 34    06                PSHS     D       *  -->  CTR2   ﾆｶｸﾉｳ
4007 CC    0040              LDD      #$0040  *ｿﾞｳﾌﾞﾝ ( 64byte )
400A 34    06                PSHS     D       *  -->  CTR    ﾆｶｸﾉｳ
400C CC    4800              LDD      #$4800  *ﾒｲﾝ.ﾊﾞｯﾌｧ ﾎﾟｲﾝﾀ
400F 34    06                PSHS     D       *  -->  MBFPTR ﾆｶｸﾉｳ
4011 CC    D800              LDD      #$D800  *ｻﾌﾞ.ｿｰｽ ﾎﾟｲﾝﾀ
4014 34    06                PSHS     D       *  -->  SUBPTR ﾆｶｸﾉｳ
4016 1F    43                TFR      S,U     *SP -->Ureg ﾆ ｺﾋﾟｰ
4018 EC    46                LDD      CTR2,U
401A C4    3F                ANDB     #$3F    *64byte ﾐﾏﾝ ﾉ ﾊｽｳ ｱﾘ ?
401C 27    03    4021        BEQ      SML1    *ﾅｹﾚﾊﾞ ｽｷｯﾌﾟ
```

```
401E 4F                       CLRA
                       *
401F ED   44           SMLOOP STD   CTR,U    *ｶｳﾝﾀ ｶｸﾉｳ
4021 EC   C4           SML1   LDD   SUBPTR,U *ｻﾌﾞ.ﾎﾟｲﾝﾀ
4023 ED   8C 2C               STD   <SMSOUC,PCR * --> RCB ﾄｼﾃ ｶｸﾉｳ
4026 8D   31    4059          BSR   SMTFR1   *ﾃﾝｿｳ ｼﾞｯｺｳ
4028 1F   43                  TFR   S,U      *ﾎﾟｲﾝﾀ ﾅﾄﾞﾉ ｺｳｼﾝ
402A EC   C4                  LDD   SUBPTR,U
402C E3   44                  ADDD  CTR,U    *ｻﾌﾞ.ﾎﾟｲﾝﾀ ﾉ ｺｳｼﾝ
402E ED   C4                  STD   SUBPTR,U
4030 EC   42                  LDD   MBFPTR,U
4032 E3   44                  ADDD  CTR,U    *ﾒｲﾝ.ﾎﾟｲﾝﾀ ﾉ ｺｳｼﾝ
4034 ED   42                  STD   MBFPTR,U
4036 EC   46                  LDD   CTR2,U
4038 A3   44                  SUBD  CTR,U    *DATAｶｳﾝﾀ ﾉ ｺｳｼﾝ
403A ED   46                  STD   CTR2,U
403C 27   05    4043          BEQ   SMRET
403E CC   0040                LDD   #64
4041 20   DC    401F          BRA   SMLOOP   *ｸﾘｶｴｼ !!!
                       *
4043 32   6A           SMRET  LEAS  10,S     *ｽﾀｯｸ ｦ ﾓﾄﾆ ﾓﾄﾞｽ
4045 39                       RTS
                       *
                       *  ｻﾌﾞｼｽﾃﾑ ﾃﾝｿｳﾃﾞｰﾀ
                       *
4046      00           SMDATA FCB   0,0,$3F *TEST ｺﾏﾝﾄﾞ
4049      59                  FCC   'YAMAUCHI' *ｷｰﾜｰﾄﾞ
4051      91                  FCB   $91      *MOVE ｻﾌﾞｺﾏﾝﾄﾞ
4052      0002         SMSOUC RMB   2        *ﾃﾝｿｳﾓﾄ ADDR
4054      D3C0                FDB   $D3C0    *ﾃﾝｿｳｻｷ ADDR
4056      0040                FDB   64       *ﾃﾝｿｳ ﾊﾞｲﾄｽｳ
4058      90                  FCB   $90      *END  ｻﾌﾞｺﾏﾝﾄﾞ
          4059         .SMD   EQU   *
                       *
                       *  ｻﾌﾞﾙｰﾁﾝ ｸﾞﾝ
                       *
4059 30   8C EA        SMTFR1 LEAX  <SMDATA,PCR *ﾃﾝｿｳ ｻﾌﾞﾙｰﾁﾝ
405C 108E FC80                LDY   #$FC80
4060 C6   13                  LDB   #.SMD-SMDATA
4062 8D   27    408B          BSR   CM_TFR
4064 8D   46    40AC          BSR   HRST
4066 8D   2D    4095          BSR   HLTWD
4068 8D   56    40C0          BSR   RDSTAT
406A 26   0F    407B          BNE   ERROR
                       *
406C 8E   FCC0         RT_TFR LDX   #SH.BUF  *ﾒｲﾝ.ﾊﾞｯﾌｧ ﾍﾉ ﾖﾐｺﾐ
406F 10AE 42                  LDY   MBFPTR,U
4072 E6   45                  LDB   CTR+1,U
4074 8D   17    408D          BSR   CMTFR2
                       *
4076 8D   3F    40B7   SM1END BSR   REDYRQ
4078 8D   32    40AC          BSR   HRST
407A 39                       RTS
                       *
407B 34   02           ERROR  PSHS  A        *ERROR ﾉ ｼｮﾘ
407D 8D   F7    4076          BSR   SM1END
407F 30   8C 43               LEAX  <ERRMSG-1,PCR
4082 BD   9BDB                JSR   PO
4085 35   02                  PULS  A
4087 BD   AC3D                JSR   OUT_A
408A 39                       RTS
                       *
408B 8D   0D    409A   CM_TFR BSR   HLT      *ｺﾏﾝﾄﾞﾉ ﾃﾝｿｳ
408D A6   80           CMTFR2 LDA   ,X+      *ｹｯｶ ﾉ ﾃﾝｿｳ
408F A7   A0                  STA   ,Y+
4091 5A                       DECB
4092 26   F9    408D          BNE   *-5
4094 39                       RTS
                       *
4095 86   18           HLTWD  LDA   #24      *HALT With Delay
4097 4A                       DECA
4098 26   FD    4097          BNE   *-1
409A B6   FD05         HLT    LDA   $FD05    *HALT Request
409D 2B   FB    409A          BMI   HLT
409F 1A   50                  ORCC  #$50
40A1 86   80                  LDA   #$80
40A3 B7   FD05                STA   $FD05
40A6 B6   FD05                LDA   $FD05
40A9 2A   FB    40A6          BPL   *-3
```

```
40AB 39                 RTS
                *
40AC 34    02   HRST    PSHS   A       *Halt RESET
40AE 4F                 CLRA
40AF B7    FD05         STA    BSYHLT
40B2 1C    AF           ANDCC  #$AF
40B4 35    02           PULS   A
40B6 39                 RTS
                *
40B7 F6    FC80 REDYRQ  LDB    SH.RAM  *READY REQUEST
40BA CA    80           ORB    #$80
40BC F7    FC80         STB    SH.RAM
40BF 39                 RTS
                *
40C0 B6    FC80 RDSTAT  LDA    $FC80   *READ STATUS
40C3 84    7F           ANDA   #$7F
40C5 39                 RTS
                *
                *  ｴﾗｰ ﾒｯｾｰｼﾞ
                *
40C6       0D0A ERRMSG  FDB    $0D0A   *Error Message
40C8       2A           FCC    '*** Error Occured ! ***  No.=$'
40E6       00           FCB    0
                *
           4000         END    $4000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40E6
PROGRAM ENTRY ADDR=4000
```

図1・2・2　プログラム例1・2・1におけるスタックの利用の仕方とラベル名の付け方



Ｉ－２－３　サブルーチンコール命令（ＪＳＲサブコマンド）

この命令は，サブシステムのメモリ上に駐在するマシン語プログラムを，サブルーチンの形で呼び出して実行させるサブコマンドです．呼び出されるマシン語サブルーチンは駐在の形状によって，

① ＲＯＭ内のプログラムのうち，サブルーチンの形をとっているもの

② 共有ＲＡＭ上のショートプログラム

③ あらかじめユーザが転送命令などによって，ＲＡＭ上に転送しておいたサブルーチンプログラム

などの場合（あるいはこれらの組合せ）が考えられますが，いずれの場合であっても，実行後にＲＴＳ，もしくはＲＴＳに相当するインストラクション（ＰＵＬＳ　ＰＣ　など）によって，ＴＥＳＴコマンド制御下のルーチンへと戻ってくるような構造であることが必要です．そうでない場合には，エラーが発生したり，場合によって暴走することもあり，実行後の正常動作は保証できません．

それでは，①～③の３つのケースに応じて，実際のプログラム例で見ていくことにします．

例Ｉ・２・２　①のケースの例で，ＲＯＭ上にあるＧＥＴＢＡ(Get Buffer Address)コマンド処理系のエントリアドレスをそのまま呼び出して，実際にＧＥＴＢＡコマンドを実行させるのと全く同様な処理を，サブシステムに行わせます．とりたてて面白味のないサンプルですが，このサンプルを見れば，どのような手順で内部サブルーチンを呼び出したらよいか，ということがよく理解できると思います．

```
                                NAM     GETBAD   *Sample Program 1.2.2
                                OPT     MEM,NOGEN * JSR to Get Buffer Address
4000                            ORG     $4000    *This Program is
                       *                         *    POSITION-INDEPENDANT
                       *
           FD05        BSYHLT   EQU     $FD05    *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
           FC80        SH.RAM   EQU     $FC80    *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
           9BDB        OUTSTR   EQU     $9BDB    *ﾓｼﾞﾚﾂ ﾉ ｼｭﾂﾘｮｸ
           AC3D        OUTA16   EQU     $AC3D    *A reg ｦ 16ｼﾝｽｳ ﾃﾞ ｼｭﾂﾘｮｸ
           D08E        OUT_A    EQU     $D08E    *1ﾓｼﾞｼｭﾂﾘｮｸ
           9B50        CR.LF.   EQU     $9B50    *ｷｬﾘｯｼﾞﾘﾀｰﾝ & ﾗｲﾝﾌｨｰﾄﾞ
                       *
                       *
           4000        START·   EQU     *        *Start of MAIN Program
4000 8D    42   4044            BSR     SUBGBA   *Get Buffer Address ｼﾞｯｺｳ
4002 34    06                   PSHS    D
4004 CC    0004                 LDD     #$0004
4007 E1    61                   CMPB    1,S
4009 25    02   400D            BLO     START1
400B C6    13                   LDB     #$13
400D ED    8D 0089     START1   STD     SBAADR,PCR
4011 8D    23   4036            BSR     SBAORD   *Set Buffer Address ｼﾞｯｺｳ
4013 30    8D 0097              LEAX    MSG-1,PCR
4017 8D    48   4061            BSR     PRTMSG   *Message ｼｭﾂﾘｮｸ
4019 A6    E4                   LDA     0,S
401B 8D    41   405E            BSR     PRTA16   *X ｻﾞﾋｮｳ ｼｭﾂﾘｮｸ
```

```
401D 86   2C                     LDA    #',
401F 8D   3A    405B             BSR    PRT_A
4021 86   24                     LDA    #'$
4023 8D   36    405B             BSR    PRT_A
4025 A6   61                     LDA    1,S
4027 8D   35    405E             BSR    PRTA16  *Y ｻﾞﾋｮｳ ｼｭﾂﾘｮｸ
4029 86   29                     LDA    #')
402B 8D   2E    405B             BSR    PRT_A
402D BD   9B50                   JSR    CR.LF.  *ｷｬﾘｯｼﾞﾘﾀｰﾝ & ﾗｲﾝﾌｨｰﾄﾞ
4030 35   06                     PULS   D
4032 ED   8D 0064                STD    SBAADR,PCR
                          *
          4036            SBAORD EQU    *       *Set Buffer Address ORDER
4036 30   8D 005B                LEAX   SBADAT,PCR
403A 108E FC80                   LDY    #SH.RAM
403E E6   8D 0052                LDB    SBALNG,PCR
4042 20   20    4064             BRA    CM_TFR
                          *
          4044            SUBGBA EQU    *       *Get Buffer Address CALL
4044 30   8D 0058                LEAX   GBADAT,PCR
4048 108E FC80                   LDY    #SH.RAM
404C E6   8D 004F                LDB    GBALNG,PCR
4050 8D   12    4064             BSR    CM_TFR
4052 8D   1B    406F             BSR    HLT     *ｹｯｶ ﾉ ﾖﾐｺﾐ
4054 FC   FC83                   LDD    SH.RAM+3
4057 8D   32    408B             BSR    REDYRQ
4059 20   26    4081             BRA    HRST
                          *
405B 7E   D08E            PRT_A  JMP    OUT_A
405E 7E   AC3D            PRTA16 JMP    OUTA16
4061 7E   9BDB            PRTMSG JMP    OUTSTR
                          *
4064 8D   09    406F      CM_TFR BSR    HLT     *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
4066 A6   80                     LDA    ,X+
4068 A7   A0                     STA    ,Y+
406A 5A                          DECB
406B 26   F9    4066             BNE    *-5
406D 20   12    4081             BRA    HRST
                          *
406F B6   FD05            HLT    LDA    $FD05   *HALT Request to ｻﾌﾞｼｽﾃﾑ
4072 2B   FB    406F             BMI    HLT     * if BUSY then Wait
4074 1A   50                     ORCC   #$50    * FIRQ,IRQ ｷﾝｼ
4076 86   80                     LDA    #$80    * BUSYﾌﾗｸﾞ ﾉ ｾｯﾄ
4078 B7   FD05                   STA    $FD05
407B B6   FD05                   LDA    $FD05   * Wait untill HALT ｱｸﾉﾘｯｼﾞ
407E 2A   FB    407B             BPL    *-3
4080 39                          RTS
                          *
4081 34   02              HRST   PSHS   A       *HALT ﾉ ｶｲｼﾞｮ
4083 4F                          CLRA
4084 B7   FD05                   STA    BSYHLT
4087 1C   AF                     ANDCC  #$AF    * FIRQ,IRQ ｷｮｶ
4089 35   82                     PULS   A,PC
                          *
408B 34   04              REDYRQ PSHS   B       *READY Request to ｻﾌﾞｼｽﾃﾑ
408D C6   80                     LDB    #$80
408F F7   FC80                   STB    SH.RAM
4092 35   84                     PULS   B,PC
                          *
4094      0A              SBALNG FCB    S.DATA-SBADAT *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
4095      0000            SBADAT FDB    $0000   *ﾃﾝｿｳﾃﾞｰﾀ
4097      03                     FCB    $03     * PUT ｺﾏﾝﾄﾞｺｰﾄﾞ
4098      06                     FCB    S.DATA-SBA1 * ﾃﾝｿｳﾊﾞｲﾄｽｳ
4099      12              SBA1   FCB    $12     * Set Buf.Addr.ORDER
409A      0004            SBAADR FDB    $0004   * Set Buf.Addr.DATA
409C      11                     FCB    $11     * Start Field ORDER
409D      07                     FCB    $07     * Attribute
409E      05                     FCB    $05     * Erase Line ORDER
          409F            S.DATA EQU    *       *End of DATA
                          *
409F      0F              GBALNG FCB    G.DATA-GBADAT *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
40A0      0000            GBADAT FDB    $0000   *ﾃﾝｿｳﾃﾞｰﾀ
40A2      3F                     FCB    $3F     * TEST ｺﾏﾝﾄﾞｺｰﾄﾞ
40A3      59                     FCC    /YAMAUCHI/ * KEYWORD
40AB      93                     FCB    $93     * JSR ｻﾌﾞｺﾏﾝﾄﾞ
40AC      ECEC                   FDB    $ECEC   * GETBAD ｴﾝﾄﾘｱﾄﾞﾚｽ
40AE      90                     FCB    $90     * END ｻﾌﾞｺﾏﾝﾄﾞ
          40AF            G.DATA EQU    *       *End of DATA
          40AF            MSG    EQU    *       *Message
40AF      110E                   FDB    $110E
```

```
11070  40B1       20              FCC     /   * Current Buffer Address = ($/
11080  40D1       00              FCB     $00
11090             4000            END     START
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40D1
PROGRAM ENTRY ADDR=4000
```

実行例

```
LOCATE 0,15


    * Current Buffer Address = ($0F,$0A)



                ABCDE




Ready
LOCATE 10,10:PRINT"ABCDE";:EXEC&H4000:
```

例1・2・3　これもまた，①のケースの例でサブシステムのROM内ルーチンを利用して，Break キーの押し離しに応じて，インサートモードLED(「INS」インジケータ）の点滅，BEEP音の発生，スクロールの実行などをリアルタイムに実行させるプログラムです．プログラムを終了させるためには，Break キー以外の任意のキーを押せばよいようになっています．本プログラムのミソは，メイン側のすべての割込みを禁止して，さらに，キーストローブがメイン側に加わるようにしている点です．ゲーム等にちょっとした色を添えたいような時に，このプログラムを応用すればよいかと思われます．

```
10010                             NAM     KEYINS  *Sample Program 1.2.3
10020                             OPT     MEM,NOGEN * INS.ｲﾝｼﾞｹｰﾀ ﾉ ﾃﾝﾒﾂ
10030  4000                       ORG     $4000   *This Program is
10040                   *                         *     POSITION-INDEPENDANT
10050                   *
10060           FD05    BSYHLT EQU    $FD05   *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
10070           FC80    SH.RAM EQU    $FC80   *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
10080                   *
10090           EAE8    INSON  EQU    $EAE8   *INS.ｲﾝｼﾞｹｰﾀ ON
10100           EAE4    INSOFF EQU    $EAE4   *INS.ｲﾝｼﾞｹｰﾀ OFF
```

```
                D033     ..INS  EQU   $D033    *ｲﾝｻｰﾄﾓｰﾄﾞﾌﾗｸﾞ
                E6A8     BELL   EQU   $E6A8    *BELL order ﾙｰﾁﾝ
                E82E     SCROLL EQU   $E82E    *Scroll ﾙｰﾁﾝ
                         *
                         *      PROGRAM  START
                         *
                4000     KEYINS EQU   *        *Start of MAIN Program
4000 1A         50              ORCC  #$50     * FIRQ,IRQ ｷﾝｼ
4002 86         01              LDA   #$01     * ｷｰIRQ ｦ ｷｮｶ
4004 B7         FD02            STA   $FD02
                         *
4007 B6         FD03     BRKWAI LDA   $FD03    *Breakｷｰ ｲｶﾞｲﾉ
400A 85         01              BITA  #$01     *  ｷｰ ｶﾞ ｵｻﾚﾃｲﾀﾗ
400C 27         1C   402A       BEQ   INSEND   *  LOOPｼｮﾘ ｦ ｼｭｳﾘｮｳ
400E B6         FD04            LDA   $FD04    *Breakｷｰ DOWN ?
4011 85         02              BITA  #$02
4013 26         F2   4007       BNE   BRKWAI
                         *
4015 30         8D 0053         LEAX  BRKON,PCR *Breakｷｰ ON
4019 8D         15   4030       BSR   CM_TFR
                         *
401B B6         FD04     BRKCNT LDA   $FD04    *Breakｷｰ UP ?
401E 85         02              BITA  #$02
4020 27         F9   401B       BEQ   BRKCNT
                         *
4022 30         8D 0058         LEAX  BRKOFF,PCR *Breakｷｰ OFF
4026 8D         08   4030       BSR   CM_TFR
4028 20         DD   4007       BRA   BRKWAI   *LOOPｼｮﾘ ﾉ ｿﾞｯｺｳ
                         *
402A 7F         FD02     INSEND CLR   $FD02    *ｷｰIRQ ｦ ﾏｽｸ
402D 1C         AF              ANDCC #$AF     *FIRQ,IRQ ﾉ ｷｮｶ
402F 39                         RTS
                         *
4030 8D         1E   4050 CM_TFR BSR  HLT      *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
4032 33         8D 002A         LEAU  TSTCMD,PCR *ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> U reg
4036 108E       FC80            LDY   #SH.RAM
403A E6         C0              LDB   ,U+
403C A6         C0       CMTFR1 LDA   ,U+
403E A7         A0              STA   ,Y+
4040 5A                         DECB
4041 26         F9   403C       BNE   CMTFR1
                         *
4043 E6         80              LDB   ,X+      *ｺﾏﾝﾄﾞﾊﾟﾗﾒｰﾀ ﾉ ﾃﾝｿｳ
4045 A6         80       CMTFR2 LDA   ,X+
4047 A7         A0              STA   ,Y+
4049 5A                         DECB
404A 26         F9   4045       BNE   CMTFR2
404C 7F         FD05     HRST   CLR   BSYHLT   *HALT ﾉ ｶｲｼﾞｮ
404F 39                         RTS
                         *
4050 B6         FD05     HLT    LDA   $FD05    *HALT Request to ｻﾌﾞｼｽﾃﾑ
4053 2B         FB   4050       BMI   HLT      * if BUSY then Wait
4055 86         80              LDA   #$80     * BUSYﾌﾗｸﾞ ｦ ｾｯﾄ
4057 B7         FD05            STA   $FD05
405A B6         FD05            LDA   $FD05    * Wait untill HALT ｱｸﾉﾘｯｼﾞ
405D 2A         FB   405A       BPL   *-3
405F 39                         RTS
                         *
4060            0B       TSTCMD FCB   T.DATA-TSTDAT *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
4061            0000     TSTDAT FDB   $0000    *ﾃﾝｿｳﾃﾞｰﾀ
4063            3F              FCB   $3F      * TEST ｺﾏﾝﾄﾞｺｰﾄﾞ
4064            59              FCC   /YAMAUCHI/ * KEYWORD
                406C     T.DATA EQU   *
                         *
406C            11       BRKON  FCB   .ONDAT-ONDATA *ﾊﾞｲﾄｽｳ
406D            93       ONDATA FCB   $93      *JSR ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
406E            EAE8            FDB   INSON    * INSON ｴﾝﾄﾘ
4070            91              FCB   $91      *MOVEｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
4071            E0C5            FDB   $E0C5    * ﾃﾞｰﾀ '$FF' ｦ ﾓﾂ ｱﾄﾞﾚｽ
4073            D033            FDB   ..INS    * ｲﾝｻｰﾄﾓｰﾄﾞﾌﾗｸﾞ
4075            0001            FDB   $0001    * ﾃﾝｿｳ ﾊﾞｲﾄｽｳ
4077            93              FCB   $93      *JSR ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
4078            E6A8            FDB   BELL     * BELL ORDER ﾙｰﾁﾝ ｴﾝﾄﾘ
407A            93              FCB   $93      *JSR ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
407B            E82E            FDB   SCROLL   * Scorll ﾙｰﾁﾝ ｴﾝﾄﾘ
407D            90              FCB   $90      *END ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
                407E     .ONDAT EQU   *
                         *
407E            0B       BRKOFF FCB   .OFDAT-OFDATA *ﾊﾞｲﾄｽｳ
```

```
407F        93         OFDATA FCB     $93       *JSR ｻﾌﾞｺﾏﾝﾄﾞ
4080        EAE4              FDB     INSOFF    * INSOFF ｴﾝﾄﾘ
4082        91                FCB     $91       *MOVE ｻﾌﾞｺﾏﾝﾄﾞ
4083        E037              FDB     $E037     * ﾃﾞｰﾀ '$00' ｦ ﾓﾂ ｱﾄﾞﾚｽ
4085        D033              FDB     ..INS     * ｲﾝｻｰﾄﾓｰﾄﾞﾌﾗｸﾞ
4087        0001              FDB     $0001     * ﾃﾝｿｳ ﾊﾞｲﾄｽｳ
4089        90                FCB     $90       *END ｻﾌﾞｺﾏﾝﾄﾞ
            408A       .OFDAT EQU     *
                       *
            4000              END     KEYINS
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=4089
PROGRAM ENTRY ADDR=4000
```

例１・２・４　先程の②のケースで，共有ＲＡＭ上に送り込んだ自前のサブルーチンをコールして，ＶＲＡＭを同じパターンの図形で埋めていくプログラムです．ＶＲＡＭへの書込みを行うため，ＶＲＡＭアクセスフラグをオンにしておく点に気をつけて下さい．

```
                                   NAM    FILL      *Sample Program 1.2.4
                                   OPT    MEM,NOGEN * FILL VRAM with Areg
4000                               ORG    $4000     *This Program is
                          *                         *    POSITION-INDEPENDANT
                          *
               FD05       BSYHLT   EQU    $FD05     *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
               FC80       SH.RAM   EQU    $FC80     *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
               9BDB       OUTSTR   EQU    $9BDB     *ﾓｼﾞﾚﾂ ﾉ ｼｭﾂﾘｮｸ
               AC3D       OUT_A    EQU    $AC3D     *A reg ｦ 16ｼﾝｽｳ ﾃﾞ ｼｭﾂﾘｮｸ
                          *
               D409       VACC     EQU    $D409     *VRAM ｱｸｾｽﾌﾗｸﾞ (ｻﾌﾞｼｽﾃﾑﾅｲ)
                          *
                          *
               4000       FILL     EQU    *         *Start of MAIN Program
4000 86        A1                  LDA    #%10100001 *VRAM ﾆ ｶｷｺﾑ ﾊﾟﾀｰﾝ
4002 A7        8D 009A             STA    PATTRN,PCR
4006 CE        8000                LDU    #$8000    *VRAM ﾉ ｾﾝﾀｸ (ﾐﾄﾞﾘ)
4009 EF        8D 009B             STU    VRAMS,PCR
400D 30        8D 007F             LEAX   TSTDAT,PCR * ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> X reg
4011 108E      FC80                LDY    #SH.RAM * ｷｮｳﾕｳRAMｾﾝﾄｳ --> Y reg
4015 E6        8D 0076             LDB    LENGTH,PCR * ﾃﾝｿｳﾊﾞｲﾄｽｳ   --> B reg
4019 8D        1D    4038          BSR    CM_TFR    * ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
401B 8D        37    4054          BSR    HRST      * HALT ﾉ ｶｲｼﾞｮ
401D 8D        23    4042          BSR    HLT       * HALT
401F 8D        47    4068          BSR    RDSTAT    * STATUS ﾉ ﾖﾐｺﾐ
4021 26        05    4028          BNE    ERROR     * ｴﾗｰ ﾅﾗ ERROR ﾍ...
                          *
4023 8D        3A    405F FILEND   BSR    REDYRQ    * ｻﾌﾞｼｽﾃﾑ ﾆ Ready Request
4025 8D        2D    4054          BSR    HRST      * HALT ｶｲｼﾞｮ
4027 39                            RTS              *End of MAIN Program
                          *
4028 34        02         ERROR    PSHS   A         *Error ﾉ ｼｮﾘ
402A 8D        F7    4023          BSR    FILEND    * Ready Request
402C 30        8C 3E               LEAX   <ERRMSG-1,PCR ｴﾗｰﾒｯｾｰｼﾞｾﾝﾄｳ --> X reg
402F BD        9BDB                JSR    OUTSTR    *     ﾒｯｾｰｼﾞ ﾉ ｼｭﾂﾘｮｸ
4032 35        02                  PULS   A
4034 BD        AC3D                JSR    OUT_A     * ERROR Number ｼｭﾂﾘｮｸ
4037 39                            RTS
                          *
4038 8D        08    4042 CM_TFR   BSR    HLT       *ｺﾏﾝﾄﾞﾉ ﾃﾝｿｳ
403A A6        80         CMTFR2   LDA    ,X+
403C A7        A0                  STA    ,Y+
403E 5A                            DECB
403F 26        F9    403A          BNE    *-5
4041 39                            RTS
                          *
4042 B6        FD05       HLT      LDA    $FD05     *HALT Request to ｻﾌﾞｼｽﾃﾑ
4045 2B        FB    4042          BMI    HLT       * if BUSY then Wait
4047 1A        50                  ORCC   #$50      * FIRQ,IRQ ｷﾝｼ
4049 86        80                  LDA    #$80      * BUSYﾌﾗｸﾞ ﾉ ｾｯﾄ
```

```
404B B7   FD05               STA    $FD05
404E B6   FD05               LDA    $FD05     * Wait untill HALT ｱｸﾉﾘｯｼﾞ
4051 2A   FB    404E         BPL    *-3
4053 39                      RTS
                     *
4054 34   02         HRST    PSHS   A         *HALT ﾉ ｶｲｼﾞｮ
4056 4F                      CLRA
4057 B7   FD05               STA    BSYHLT
405A 1C   AF                 ANDCC  #$AF      * FIRQ,IRQ ｷｮｶ
405C 35   02                 PULS   A
405E 39                      RTS
                     *
405F F6   FC80       REDYRQ  LDB    SH.RAM    *READY Request to ｻﾌﾞｼｽﾃﾑ
4062 CA   80                 ORB    #$80
4064 F7   FC80               STB    SH.RAM
4067 39                      RTS
                     *
4068 B6   FC80       RDSTAT  LDA    $FC80     *Read STATUS
406B 84   7F                 ANDA   #$7F
406D 39                      RTS
                     *
406E      0D0A       ERRMSG  FDB    $0D0A     *Error Message
4070      2A                 FCC    '*** Error Occured ! ***  No.=$'
408E      00                 FCB    0
                     *
408F      26         LENGTH  FCB    T.DATA-TSTDAT *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
4090      0000       TSTDAT  FDB    $0000     *ﾃﾝｿｳﾃﾞｰﾀ
4092      3F                 FCB    $3F       * TEST ｺﾏﾝﾄﾞｺｰﾄﾞ
4093      59                 FCC    /YAMAUCHI/ * KEYWORD
409B      93                 FCB    $93       * JSR ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
409C      D38F               FDB    TSTSUB-TSTDAT+$D380 * Entry Addr.
409E      90                 FCB    $90       * END ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
                     *
          409F       TSTSUB  EQU    *         * ﾏｼﾝｺﾞｻﾌﾞﾙｰﾁﾝ
409F      86                 FCB    $86       * LDA #
40A0      0001       PATTRN  RMB    1         * Pattern
40A1 B5   D409               BITA   VACC      * ｱｸｾｽﾌﾗｸﾞON
40A4 8E   3E80               LDX    #16000
40A7      CE                 FCB    $CE       * LDU #
40A8      0002       VRAMS   RMB    2         * VRAM selection
40AA 1F   89                 TFR    A,B
40AC ED   C1         STLOOP  STD    ,U++      * Fill VRAM with Areg
40AE 30   1E                 LEAX   -2,X
40B0 26   FA    40AC         BNE    STLOOP
40B2 B7   D409               STA    VACC      * ｱｸｾｽﾌﾗｸﾞOFF
40B5 39                      RTS
                     *
          40B6       T.DATA  EQU    *         *End of DATA
                     *
          4000               END    FILL
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40B5
PROGRAM ENTRY ADDR=4000
```

例１・２・５　先程の③のケースで，転送命令であらかじめ補助ワークエリア上に転送しておいた自前のプログラムを呼び出して，ワークエリアの変数・フラグ類（＄Ｄ０００～＄Ｄ０ＦＦ）を１２８バイトずつメイン側に転送するプログラムです．ＢＵＳＹフラグとキャンセル割込みＩＲＱ（詳しくは第２章で説明します）でメイン↔サブ間の同期をとっていることに注目して下さい．

```
                                NAM    RDWORK  *Sample Program 1.2.5
                                OPT    MEM,NOGEN ｻﾌﾞｼｽﾃﾑ ﾜｰｸｴﾘｱ ﾉ ﾖﾐｺﾐ
4000                            ORG    $4000   *This Program is
                        *                      *    POSITION-INDEPENDANT
                        *
            FD05        BSYHLT EQU     $FD05   *BUSYﾌﾗｸﾞ,HALT & CANCEL REQ.
            FC80        SH.RAM EQU     $FC80   *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
                        *
            0040        CANCEL EQU     %01000000 *CANCEL ﾖｳｷｭｳ ﾋﾞｯﾄ
                        *
            0000               SETDP   $00     *ﾒｲﾝｼｽﾃﾑﾅｲﾃﾞ ﾉ DP ﾉ ｱﾀｲ
                        *
                        *      MAIN  Program
                        *
            4000        RDWORK EQU     *       *Start of MAIN Program
4000 1A     50                 ORCC    #$50    *FIRQ,IRQ ｷﾝｼ
4002 30     8D 004C            LEAX    TSTDAT,PCR * ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> X reg
4006 108E   FC80               LDY     #SH.RAM *      ｷｮｳﾕｳRAMｾﾝﾄｳ --> Y reg
400A E6     8D 0043            LDB     LENGTH,PCR *   ﾃﾝｿｳﾊﾞｲﾄｽｳ    --> B reg
400E 8D     23    4033         BSR     CM_TFR  *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
4010 8D     3B    404D         BSR     HRST    *HALT ﾉ ｶｲｼﾞｮ
4012 C6     02                 LDB     #2      * ﾃﾝｿｳ ｶｲｽｳ
4014 34     04                 PSHS    B       *      --> ｽﾀｯｸ ﾆ ｶｸﾉｳ
4016 108E   5000               LDY     #$5000  * ﾒｲﾝｶﾞﾜ ﾊﾞｯﾌｧｾﾝﾄｳ --Y reg
401A 8E     FC80        RDWOR1 LDX     #SH.RAM *ﾃﾝｿｳ ｼﾞｯｺｳ ﾙｰﾌﾟ
401D C6     80                 LDB     #128    *( ｲｯｶｲ ﾉ ﾃﾝｿｳﾌﾞﾝ ﾊ 128ﾊﾞｲﾄ )
401F 8D     12    4033         BSR     CM_TFR  *HALT & ﾃﾞｰﾀ ﾖﾐｺﾐ
4021 8D     2A    404D         BSR     HRST    *HALT ﾉ ｶｲｼﾞｮ
4023 86     40                 LDA     #CANCEL *CANCELﾜﾘｺﾐ ｦ
4025 B7     FD05               STA     BSYHLT  *    ｻﾌﾞｼｽﾃﾑ ﾆ ﾖｳｷｭｳ
4028 7F     FD05               CLR     BSYHLT  *( ﾜﾘｺﾐﾗｯﾁ ｦ ﾓﾄ ﾆ ﾓﾄﾞｽ )
                        * CANCELﾜﾘｺﾐ ﾉ ｶｹｶﾀ ﾆ ﾁｭｳｲ !!!
                        *
402B 6A     E4                 DEC     ,S      *ｶｳﾝﾀ ﾃﾞｸﾘﾒﾝﾄ
402D 26     EB    401A         BNE     RDWOR1  *  0 ﾃﾞﾅｹﾚﾊﾞ ﾙｰﾌﾟﾉｾﾝﾄｳ ﾍ
402F 1C     AF                 ANDCC   #$AF    *FIRQ,IRQ ｷｮｶ
4031 35     84                 PULS    PC,B    *End of MAIN Program
                        *
                        *      Subroutines
                        *
4033 8D     08    403D  CM_TFR BSR     HLT     *ｺﾏﾝﾄﾞ & ﾃﾞｰﾀ ﾉ ﾃﾝｿｳ
4035 A6     80                 LDA     ,X+
4037 A7     A0                 STA     ,Y+
4039 5A                        DECB
403A 26     F9    4035         BNE     *-5
403C 39                        RTS
                        *
403D B6     FD05        HLT    LDA     $FD05   *HALT Request to ｻﾌﾞｼｽﾃﾑ
4040 2B     FB    403D         BMI     HLT     * if BUSY then Wait
4042 86     80                 LDA     #$80    * BUSYﾌﾗｸﾞ ﾉ ｾｯﾄ
4044 B7     FD05               STA     $FD05
4047 B6     FD05               LDA     $FD05   * Wait untill HALT ｱｸﾉﾘｯｼﾞ-
404A 2A     FB    4047         BPL     *-3
404C 39                        RTS
                        *
404D 7F     FD05        HRST   CLR     BSYHLT  *HALT ﾉ ｶｲｼﾞｮ
4050 39                        RTS
                        *
                        *      TESTｺﾏﾝﾄﾞﾃﾝｿｳﾃﾞｰﾀ
                        *
4051        51          LENGTH FCB     ENDATA-TSTDAT *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
4052        0000        TSTDAT FDB     $0000   *ﾃﾝｿｳﾃﾞｰﾀ
4054        3F                 FCB     $3F     *     TEST ｺﾏﾝﾄﾞｺｰﾄﾞ
4055        59                 FCC     /YAMAUCHI/ * KEYWORD
405D        91                 FCB     $91     *     MOVEｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
405E        D396               FDB     SBPROG-TSTDAT+$D380 * ﾃﾝｿｳﾓﾄ Addr.
4060        D200               FDB     $D200   *              ﾃﾝｿｳｻｷ Addr.
4062        003B               FDB     ENDATA-SBPROG *        ﾃﾝｿｳﾊﾞｲﾄｽｳ
4064        93                 FCB     $93     * JSR ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
4065        D200               FDB     $D200   * JSR ﾌﾞﾝｷｻｷ Addr.
4067        90                 FCB     $90     * END ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
                        *            ｼﾞﾂﾊ ｼｮﾘｺﾞ ﾁｮｸｾﾂ CMDCTL ﾍﾄ
                        *              ｼﾞｬﾝﾌﾟｽﾙ ﾉﾃﾞ ENDｻﾌﾞｺﾏﾝﾄﾞ ﾊ ﾌﾖｳ !!!
                        *
                        *      $D200 ｲｶﾆ ｵｸﾗﾚﾙ ﾌﾟﾛｸﾞﾗﾑ ﾆ ﾀｲｽﾙ ﾗﾍﾞﾙ
                        *
            D000        ..ABRT EQU     $D000   *ｱﾎﾞｰﾄ(CANCEL)ﾌﾗｸﾞ
            D00A        .TMAC  EQU     $D00A   *ﾀｲﾏｱｸｾｽﾌﾗｸﾞ
            D380        ERRREG EQU     $D380   *ｷｮｳﾕｳRAM ｾﾝﾄｳｱﾄﾞﾚｽ
```

```
                  D381      STATUS EQU    $D381    *STATUSﾌﾗｸﾞ (ｷｮｳﾕｳRAMｼﾞｮｳ)
                  D40A      BUSYC  EQU    $D40A    *BUSYﾌﾗｸﾞ ｾｲｷﾞｮﾎﾟｰﾄ
                  E034      CMDCTL EQU    $E034    *ｺﾏﾝﾄﾞﾊﾝﾄﾞﾗ ｾﾝﾄｳ
                  E44A      MOVE   EQU    $E44A    *ﾌﾞﾛｯｸﾃﾝｿｳ ｻﾌﾞﾙｰﾁﾝ
                            *
                  00D0             SETDP  $D0      *ｻﾌﾞｼｽﾃﾑﾅｲﾃﾞﾉ DP ﾉ ｱﾀｲ
                            *
                            *      ｻﾌﾞｼｽﾃﾑ ﾉ $D200 ｲｶﾆ ｵｸﾗﾚﾙ ﾌﾟﾛｸﾞﾗﾑ
                            *
                  4068      SBPROG EQU    *        *ﾏｼﾝｺﾞｻﾌﾞﾙｰﾁﾝｾﾝﾄｳ
      4068 1A     40               ORCC   #$40     * FIRQ ｷﾝｼ
      406A 1C     EF               ANDCC  #$EF     * IRQ  ｷｮｶ
      406C B7     D40A             STA    BUSYC    * BUSYﾌﾗｸﾞON  ( Busy  )
      406F C6     02               LDB    #2       * ﾃﾝｿｳ ｶｲｽｳ
      4071 34     04               PSHS   B        *    -->･ｽﾀｯｸ ﾆ ｶｸﾉｳ
      4073 D7     0A               STB    .TMAC    * ﾀｲﾏｱｸｾｽﾌﾗｸﾞON ( NMIﾏｽｸ )
      4075 8E     D000             LDX    #$D000   * ﾃﾝｿｳｶｲｼ ｱﾄﾞﾚｽ  --> X reg
      4078 108E   D380      SBPR01 LDY    #ERRREG  * ｷｮｳﾕｳRAM ｾﾝﾄｳ  --> Y reg
      407C C6     80               LDB    #128     * 1ｶｲﾉﾃﾝｿｳﾊﾞｲﾄｽｳ --> B reg
      407E D7     0A               STB    .TMAC    * ﾀｲﾏｱｸｾｽﾌﾗｸﾞON
      4080 BD     E44A             JSR    MOVE     * ﾌﾞﾛｯｸﾃﾝｿｳ
      4083 B5     D40A             BITA   BUSYC    * BUSYﾌﾗｸﾞOFF ( Ready )
      4086 C6     FF               LDB    #$FF
      4088 D7     0A        SBPR02 STB    .TMAC    * ﾀｲﾏｱｸｾｽﾌﾗｸﾞON
      408A 96     00               LDA    ..ABRT   * ｱﾎﾞｰﾄﾌﾗｸﾞ  ﾃｽﾄ
      408C 27     FA   4088        BEQ    SBPR02   * ﾜﾘｺﾐ ｶﾞ ｶｶﾙﾏﾃﾞ ﾙｰﾌﾟ
      408E D7     0A               STB    .TMAC    * ﾀｲﾏｱｸｾｽﾌﾗｸﾞON
      4090 0F     00               CLR    ..ABRT   * ｱﾎﾞｰﾄﾌﾗｸﾞ  ｸﾘｱ
      4092 6A     E4               DEC    ,S       * ｶｳﾝﾀ ﾃﾞｸﾘﾒﾝﾄ
      4094 26     E2   4078        BNE    SBPR01   * 0 ﾃﾞﾅｹﾚﾊﾞ ｸﾘｶｴｼ
      4096 7F     D380             CLR    ERRREG   * ｴﾗｰｺｰﾄﾞﾚｼﾞｽﾀ ｸﾘｱ
      4099 7F     D381             CLR    STATUS   * STATUS ﾚｼﾞｽﾀ ｸﾘｱ
      409C 0F     0A               CLR    .TMAC    * ﾀｲﾏｱｸｾｽﾌﾗｸﾞ  ｸﾘｱ
      409E 1C     AF               ANDCC  #$AF     * FIRQ,IRQ ｷｮｶ
      40A0 7E     E034             JMP    CMDCTL   * ｺﾏﾝﾄﾞﾊﾝﾄﾞﾗｾﾝﾄｳ  ﾆ ｼﾞｬﾝﾌﾟ
                            *
                  40A3      ENDATA EQU    *        *End of DATA
                            *
                  4000             END    RDWORK
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40A2
PROGRAM ENTRY ADDR=4000
```

Ｉ－２－４　移動命令（ＪＭＰサブコマンド）

ＴＥＳＴコマンドをサブシステムに実行させる際，共有ＲＡＭをフルに使いたいケースがあります．このような場合には，一連のサブコマンドリストを転送命令を用いて別の場所に移しておき，その後この移動命令によって制御をその場所に移し，サブコマンドリストを実行すると，都合よく実行させることができます．あくまでも，サブコマンドリストの参照アドレスを変えるための命令，すなわち，サブコマンド読込みポインタの値を変える命令であって，指定したアドレスのマシン語プログラムにジャンプするという意味ではありません．くれぐれも注意して下さい．

例Ｉ・２・６　コンソールバッファの内容を125バイトずつメイン側に転送します．

```
                                NAM    NSMTFR  *Sample Program 1.2.6
                                OPT    MEM,NOGEN NEW SUB to MAIN Trans.
4000                            ORG    $4000   *This Program is
                        *                      *    POSITION-INDEPENDANT
                        *
                        *  ｽﾀｯｸｼﾞｮｳ ﾉ ﾜｰｸｴﾘｱ ﾉ ﾃｲｷﾞ
                        *
             0000       SUBPTR EQU     $0000   *SUB ｶﾞﾜ ﾎﾟｲﾝﾀ
             0002       MBFPTR EQU     $0002   *MAINｶﾞﾜ ﾎﾟｲﾝﾀ
             0004       CTR    EQU     $0004   *ｿﾞｳﾌﾞﾝ
             0006       CTR2   EQU     $0006   *DATA ｶｳﾝﾀ
                        *
                        *  ﾒｲﾝ･ｻﾌﾞ ｲﾝﾀﾌｪｰｽ
                        *
             FD05       BSYHLT EQU     $FD05   *ﾒｲﾝ･ｻﾌﾞ ｲﾝﾀﾌｪｰｽ
             FC80       SH.RAM EQU     $FC80   *ｷｮｳﾕｳRAM ｾﾝﾄｳｱﾄﾞﾚｽ
                        *
                        *  ﾒｲﾝROM ﾅｲﾌﾞﾙｰﾁﾝ
                        *
             9BDB       OUTSTR EQU     $9BDB   *ﾓｼﾞﾚﾂﾋｮｳｼﾞ
             AC3D       OUT_A  EQU     $AC3D   *Areg ﾋｮｳｼﾞ
                        *
                        *  ﾌﾟﾛｸﾞﾗﾑ
                        *
             4000       NSMTFR EQU     *       *ﾒｲﾝ ﾙｰﾁﾝ
4000 CC      0FA0              LDD     #4000   *ﾃﾝｿｳﾊﾞｲﾄｽｳ ( 10K )
4003 34      06                PSHS    D       *  -->  CTR2  ﾆｶｸﾉｳ
4005 CC      007D              LDD     #125    *ｿﾞｳﾌﾞﾝ ( 125byte )
4008 34      06                PSHS    D       *  -->  CTR   ﾆｶｸﾉｳ
400A CC      5000              LDD     #$5000  *ﾒｲﾝ.ﾊﾞｯﾌｧ ﾎﾟｲﾝﾀ
400D 34      06                PSHS    D       *  -->  MBFPTR ﾆｶｸﾉｳ
400F CC      C000              LDD     #$C000  *ｻﾌﾞ.ｿｰｽ ﾎﾟｲﾝﾀ
4012 34      06                PSHS    D       *  -->  SUBPTR ﾆｶｸﾉｳ
4014 1F      43                TFR     S,U     *SP -->Ureg ﾆ ｺﾋﾟｰ
                        *
4016 EC      C4         SMLOOP LDD     SUBPTR,U *ｻﾌﾞ.ﾎﾟｲﾝﾀ
4018 ED      8C 31             STD     <SMSOUC,PCR * --> RCB ﾄｼﾃ ｶｸﾉｳ
401B 8D      36   4053         BSR     SMTFR1  *ﾃﾝｿｳ ｼﾞｯｺｳ
401D 1F      43                TFR     S,U     *ﾎﾟｲﾝﾀ ﾅﾄﾞﾉ ｺｳｼﾝ
401F EC      C4                LDD     SUBPTR,U
4021 E3      44                ADDD    CTR,U   *ｻﾌﾞ.ﾎﾟｲﾝﾀ ﾉ ｺｳｼﾝ
4023 ED      C4                STD     SUBPTR,U
4025 EC      42                LDD     MBFPTR,U
4027 E3      44                ADDD    CTR,U   *ﾒｲﾝ.ﾎﾟｲﾝﾀ ﾉ ｺｳｼﾝ
4029 ED      42                STD     MBFPTR,U
402B EC      46                LDD     CTR2,U
402D A3      44                SUBD    CTR,U   *DATAｶｳﾝﾀ ﾉ ｺｳｼﾝ
402F ED      46                STD     CTR2,U
4031 26      E3   4016         BNE     SMLOOP  *ｸﾘｶｴｼ !!!
                        *
4033 32      68         SMRET  LEAS    8,S     *ｽﾀｯｸ ｦ ﾓﾄﾆ ﾓﾄﾞｽ
4035 39                        RTS
                        *
                        *  ｻﾌﾞｼｽﾃﾑ ﾃﾝｿｳﾃﾞｰﾀ
                        *
4036         00         SMDATA FCB     0,0,$3F *TEST ｺﾏﾝﾄﾞ
4039         59                FCC     'YAMAUCHI' *ｷｰﾜｰﾄﾞ
                        *  ｻﾌﾞｺﾏﾝﾄﾞﾚﾂ ｦ ｱﾗｶｼﾞﾒ ﾎｼﾞｮﾜｰｸｴﾘｱ ﾆ ﾃﾝｿｳ
4041         91                FCB     $91     *MOVE ｻﾌﾞｺﾏﾝﾄﾞ
4042         D395              FDB     SUBLST-SMDATA+$D380
4044         D100              FDB     $D100   *ﾎｼﾞｮﾜｰｸｴﾘｱ ﾉ ｲﾁﾌﾞ ｦ ｼﾖｳ
4046         0008              FDB     ENDSMD-SUBLST *ﾊﾞｲﾄｽｳ
                        *  ｲﾄﾞｳﾒｲﾚｲ ﾉ ｼﾖｳ ﾚｲ
4048         92                FCB     $92     *JMP ｻﾌﾞｺﾏﾝﾄﾞ (ﾎﾟｲﾝﾀ ﾉ ｲﾄﾞｳ)
4049         D100              FDB     $D100   * JMP ｱﾄﾞﾚｽ
                        *  ｲｶ ﾊ ｲﾄﾞｳｻﾚﾙ ｻﾌﾞｺﾏﾝﾄﾞﾚﾂ
404B         91         SUBLST FCB     $91     *MOVE ｻﾌﾞｺﾏﾝﾄﾞ
404C         0002       SMSOUC RMB     2       *ﾃﾝｿｳﾓﾄ ADDR
404E         D383              FDB     $D383   *ﾃﾝｿｳｻｷ ADDR
4050         007D              FDB     125     *ﾃﾝｿｳ ﾊﾞｲﾄｽｳ
4052         90                FCB     $90     *END  ｻﾌﾞｺﾏﾝﾄﾞ
             4053       ENDSMD EQU     *
                        *
                        *  ｻﾌﾞﾙｰﾁﾝ ｸﾞﾝ
                        *
4053 30      8C E0      SMTFR1 LEAX    <SMDATA,PCR *ﾃﾝｿｳ ｻﾌﾞﾙｰﾁﾝ
4056 108E    FC80              LDY     #$FC80
405A C6      1D                LDB     #ENDSMD-SMDATA
405C 8D      27   4085         BSR     CM_TFR
```

```
405E 8D   41    40A1        BSR    HRST
4060 8D   2D    408F        BSR    HLT
4062 8D   4C    40B0        BSR    RDSTAT
4064 26   0F    4075        BNE    ERROR
                      *
4066 8E   FC83        RT_TFR LDX   #$FC83   *ﾒｲﾝ.ﾊﾞｯﾌｧ ﾍﾉ ﾖﾐｺﾐ
4069 10AE 42                LDY    MBFPTR,U
406C E6   45                LDB    CTR+1,U
406E 8D   17    4087        BSR    CMTFR2
                      *
4070 8D   35    40A7  SM1END BSR   REDYRQ
4072 8D   2D    40A1        BSR    HRST
4074 39                     RTS
                      *
4075 34   02          ERROR  PSHS  A        *ERROR ﾉ ｼｮﾘ
4077 8D   F7    4070        BSR    SM1END
4079 30   8C 39             LEAX   <ERRMSG-1,PCR
407C BD   9BDB              JSR    OUTSTR
407F 35   02                PULS   A
4081 BD   AC3D              JSR    OUT_A
4084 39                     RTS
                      *
4085 8D   08    408F  CM_TFR BSR   HLT      *ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
4087 A6   80          CMTFR2 LDA   ,X+      *ｹｯｶ ﾉ ﾃﾝｿｳ
4089 A7   A0                STA    ,Y+
408B 5A                     DECB
408C 26   F9    4087        BNE    *-5
408E 39                     RTS
                      *
408F B6   FD05        HLT    LDA   $FD05    *HALT Request
4092 2B   FB    408F        BMI    HLT
4094 1A   50                ORCC   #$50
4096 86   80                LDA    #$80
4098 B7   FD05              STA    $FD05
409B B6   FD05              LDA    $FD05
409E 2A   FB    409B        BPL    *-3
40A0 39                     RTS
                      *
40A1 7F   FD05        HRST   CLR   BSYHLT   *Halt RESET
40A4 1C   AF                ANDCC  #$AF
40A6 39                     RTS
                      *
40A7 F6   FC80        REDYRQ LDB   SH.RAM   *READY REQUEST
40AA CA   80                ORB    #$80
40AC F7   FC80              STB    SH.RAM
40AF 39                     RTS
                      *
40B0 B6   FC80        RDSTAT LDA   $FC80    *READ STATUS
40B3 84   7F                ANDA   #$7F
40B5 39                     RTS
                      *
                      *  ｴﾗｰ ﾒｯｾｰｼﾞ
                      *
40B6      0D0A        ERRMSG FDB   $0D0A    *Error Message
40B8      2A                FCC    '*** Error Occured ! ***  No.=$'
40D6      00                FCB    0
                      *
          4000              END    NSMTFR
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40D6
PROGRAM ENTRY ADDR=4000
```

1－2－5　終了命令（ENDサブコマンド）

文字どおり，TESTコマンドを終了させて，メイン側からのコマンド待ち状態へと制御を戻すためのサブコマンドです．サブコマンドリストが長い場合には，このENDサブコマンドを忘れがちですが，これを忘れると異常終了となりますから，サブコマンドリストの最後に，締めくくりとして必ず入れるようにして下さい．それでは，TESTコマンドの説明の締めくくりとして，この処理系のソースリストを掲載します．特に難解なところはないと思います．むしろ移動命令の様子などが一目でわかるのではないでしょうか．

```
                                 NAM    TEST     TEST ｺﾏﾝﾄﾞﾙｰﾁﾝ
                                 OPT    NOMEM
FF6F                             ORG    $FF6F
            00D0                 SETDP  $D0
            D38B         TSTSUB  EQU    $D38B    *ｻﾌﾞｺﾏﾝﾄﾞﾘｽﾄ ｾﾝﾄｳ
            E0ED         ERR70   EQU    $E0ED
                         *
FF6F 8E     D38B         TEST    LDX    #TSTSUB  *TEST ｺﾏﾝﾄﾞｴﾝﾄﾘ
FF72 A6     80           TSTCMD  LDA    ,X+      *ｻﾌﾞｺﾏﾝﾄﾞ ﾖﾐｺﾐ
FF74 80     90                   SUBA   #$90
FF76 2B     0B    FF83           BMI    TSTERR
FF78 27     0C    FF86           BEQ    TSTRTS   *ｻﾌﾞｺﾏﾝﾄﾞ END
FF7A 4A                          DECA
FF7B 27     18    FF95           BEQ    TSTMOV   *ｻﾌﾞｺﾏﾝﾄﾞ MOVE
FF7D 4A                          DECA
FF7E 27     07    FF87           BEQ    TSTJMP   *ｻﾌﾞｺﾏﾝﾄﾞ JMP
FF80 4A                          DECA
FF81 27     08    FF8B           BEQ    TSTJSR   *ｻﾌﾞｺﾏﾝﾄﾞ JSR
                         *
FF83 7E     E0ED         TSTERR  JMP    ERR70    *Invalid Subcommand
                         *
FF86 39                  TSTRTS  RTS             *Return to MAIN LOOP
                         *
FF87 AE     84           TSTJMP  LDX    ,X       *ｲﾄﾞｳ ｱﾄﾞﾚｽ ﾖﾐｺﾐ
FF89 20     E7    FF72           BRA    TSTCMD
                         *
FF8B EE     81           TSTJSR  LDU    ,X++     *ﾏｼﾝｺﾞｻﾌﾞﾙｰﾁﾝｺｰﾙ
FF8D 34     10                   PSHS   X        * ﾎﾟｲﾝﾀ ｾｰﾌﾞ
FF8F AD     C4                   JSR    ,U
FF91 35     10                   PULS   X
FF93 20     DD    FF72           BRA    TSTCMD
                         *
FF95 10AE   81           TSTMOV  LDY    ,X++     *Source Addr.
FF98 EE     81                   LDU    ,X++     *Destination Addr.
FF9A EC     81                   LDD    ,X++     *Byte Count
FF9C 27     D4    FF72           BEQ    TSTCMD
FF9E 34     10                   PSHS   X
FFA0 1F     01                   TFR    D,X
FFA2 A6     A0           TSTMLP  LDA    ,Y+      *ﾌﾞﾛｯｸﾃﾝｿｳ
FFA4 A7     C0                   STA    ,U+
FFA6 30     1F                   LEAX   -$01,X
FFA8 26     F8    FFA2           BNE    TSTMLP
FFAA 35     10                   PULS   X
FFAC 20     C4    FF72           BRA    TSTCMD
                         *
                                 END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=FF6F
PROGRAM END   ADDR=FFAD
PROGRAM ENTRY ADDR=****
```

第2章

サブシステムのアーキテクチャ

ハードウェアとのかかわり——

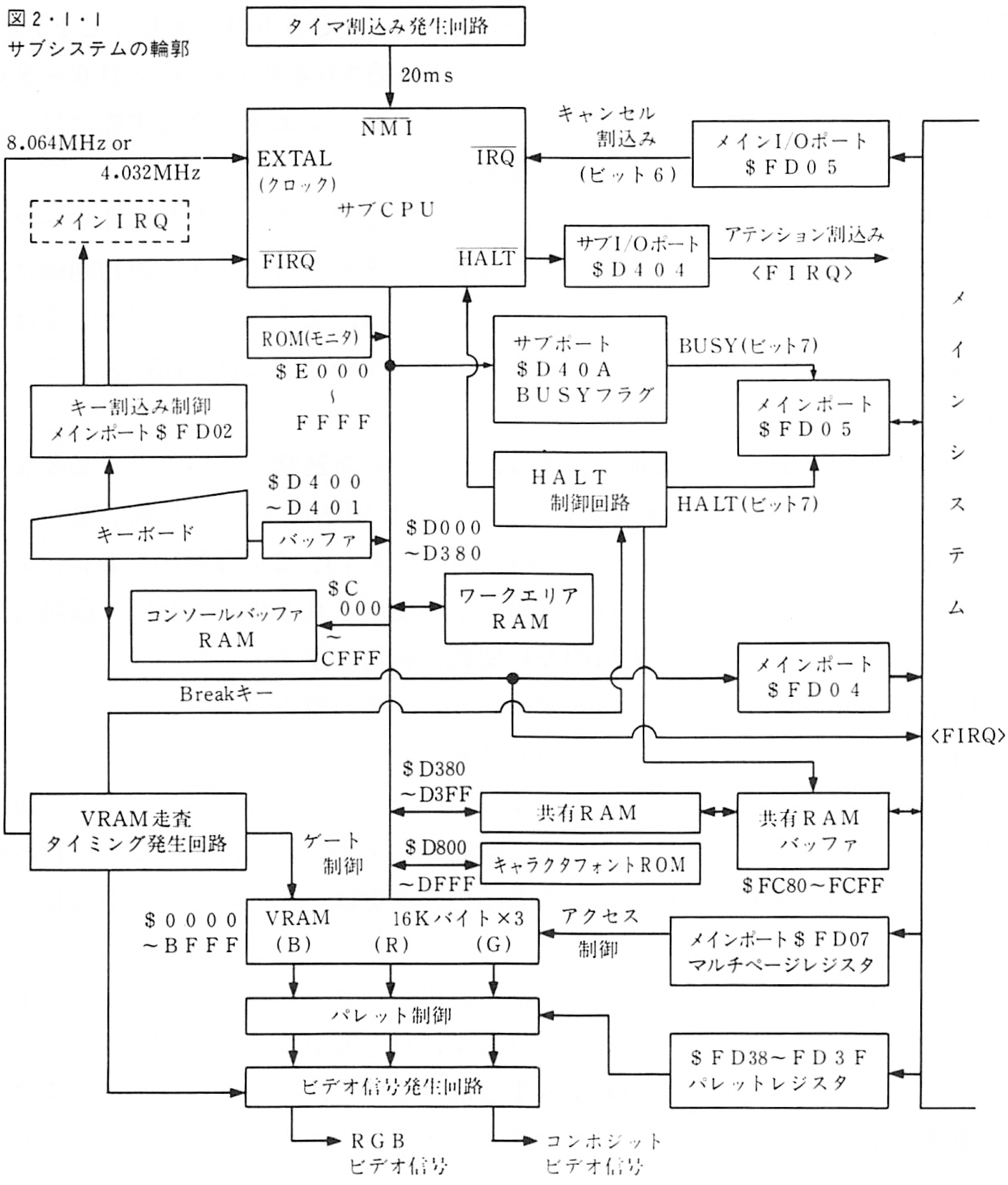
どのようなコンピュータを使いこなすにせよ，ハードウェアの輪郭がしっかりと理解できていなければ，ソフトウェアを組む上で必ずなんらかの支障をきたすことになります．このことは，サブシステムをメイン側とは独立した一個のコンピュータとして扱おうと考えていらっしゃる読者のみなさんにとって，事情はなおさらのことです．
この第2章では，サブシステムのハードウェアの輪郭，メイン↔サブ間インタフェースの模様など，ハードウェア絡みの部分を中心に解説していきます．

PHASE III
SUBSYSTEM

2－1　サブシステムの基本構造

サブシステム全体の構造の概略を描いたブロック図が，図2・1・1です．68系CPUでは，メモリマップドI／O方式が用いられているため，I／Oポートという概念は稀薄なのですが，FM－7の場合，メイン側の＄FD××領域およびサブ側の＄D400～D40Fの領域は，まさにI／Oポートそのものであり，この言葉を用いた方がわかり易いと思われるので，以後この言葉を使用することにします．図2・1・1では，メイン側はサブシステムと密接な関係にあるI／Oポートのみを示してあります．

図2・1・1
サブシステムの輪郭



2－2　サブシステムのメモリマップ

ＦＭ－７のシステムにおけるメモリ空間の割付けの様子を図２・２・１に示します．アドレス順に見ていきますと，＄００００番地から１６Ｋバイトずつ青，赤，緑とＶＲＡＭの空間が割り当てられています．

この空間をアクセスする際は，ＶＲＡＭアクセスフラグを立てておく（Ｉ／Ｏポート＄Ｄ４０９を Read)必要があります．この辺の事情については後で解説します．なお，ＶＲＡＭアクセスフラグを立ててからＶＲＡＭを実際にアクセスするまでに，１マシンサイクル以上の間をおく必要があります．

＄Ｃ０００～＄ＣＦ９Ｆ番地までの４０００バイトは，コンソールバッファの空間で，表示文字のキャラクタコードが入る部分（＄Ｃ０００～＄Ｃ７ＣＦ）と，それらの文字の属性，表示色やフィールドなどが格納される部分（＄Ｃ７Ｄ０～＄ＣＦ９Ｆ）との２つの部分からなっています．スクリーンエディタは実際には，このコンソールバッファに対して作用します．

＄ＣＦＡ０～＄ＣＦＦＦ番地までは，ハードウェアスタックとしてリザーブされています．そのため，サブシステムのＳＰ（スタックポインタ）の初期値は，＄Ｄ０００に設定されます．９６バイトというのは，スタックエリアとしては小さい感じがしますが，あくまでもサブシステムですから，これで間に合ってしまうのです．

＄Ｄ０００～＄Ｄ０９Ｆ番地の領域は，システムを運営していく上で必要なフラグ･レジスタ類が収められる変数エリアとして使用されます（「変数」といっても，ＢＡＳＩＣで使用される変数とは趣を異にします）．この領域の変数は，システム全域に渡って等価な意味を持つものとして使用されます．いわば，高級言語でいうところのグローバル（大域的な）変数に相当します．

例えば，表示色カラーコード（厳密にいうと，その時点のアトリビュートコード，第３章参照）が格納される，＄Ｄ０２Ｅというワークエリア上の変数を「赤（＝＄０２)」に書き換えてやることによって，設定されていたフィールドの属性などにいっさいお構いなしに，その次の瞬間から文字の表示はすべて赤で行われます．この＄Ｄ０２Ｅの書換えによる表示色の変更は，次のフィールド設定（ＳＦオーダ）等があるまで有効です．

＄Ｄ０２Ｅの書換えによる表示色変換を流用したサンプルが，例２・２・１です．この技法によりフィールド設定を行わずに色を変更することができるため，ＬＩＳＴ出力に用いても通常のＬＩＳＴと同じように，スクリーンエディタによって編集を行うことができます．カラフルでオリジナルなＬＩＳＴを出してみよ

図2・2・1　FM—7．サブシステムのメモリ空間の割付け

アドレス	内容	
＄0000	VRAM ［Blue］ (16Kバイト)	
＄4000	VRAM ［Red］ (16Kバイト)	
＄8000	VRAM ［Green］ (16Kバイト)	
＄C000	コンソールバッファ (4000バイト)	キャラクタコードのバッファ［文字コードが格納される］
＄C7D0		アトリビュートバッファ　［文字の属性が格納される］
＄CFA0	ハードウェアスタック　(96バイト)	スタックの方向 ↑ ↑ ↑
＄D000	※ワークエリア［サブシステムに必要なフラグ，変数類］(160バイト)	
＄D0A0	※補助ワークエリア［ローカル変数，第2スタックなどに使用］(544バイト)	
＄D2C0	PFキー定義テーブル　(160バイト)	
＄D360	キー入力バッファ　(32バイト)	
＄D380	共有RAM (128バイト)　;　メイン側のロケーションは＄FC80～＄FCFF	
＄D400	I/O領域 (1Kバイト)	実際にI/Oポートとして使用される空間（次節参照）
＄D410		《フルデコードされていないために残った未使用空間》
＄D800	キャラクタフォント　ROM　［文字コード順］(2Kバイト)	
＄E000	※モニタ　プログラム　ROM　(8Kバイト)［サブシステム全体を制御しているプログラム］	
		＄FFF6～F7　FIRQベクトル　：　＄FDAC
		＄FFF8～F9　IRQ　ベクトル　：　＄E06E
		＄FFFA～FB　SWI (Don't care)　：　＄E000
		＄FFFC～FD　NMI　ベクトル　：　＄FEBF
		＄FFFE～FF　RESETベクトル　：　＄E000

※　ワークエリア・補助ワークエリア・モニタプログラムの詳細は，第3章および第5章に掲載してあります．

うと思われる読者は，この技法を思う存分に利用して下さい．

例２・２・１　テキスト中にコロンを発見すると，そこから色を変えて出力する「オリジナルＬＩＳＴ」プログラム

```
                               NAM     CLLIST  *Sample Program 2.2.1
                               OPT     MEM,NOGEN Colorful LIST (ｶﾗﾌﾙﾘｽﾄ)
4000                           ORG     $4000   *This Program is
                        *                      *    POSITION-INDEPENDANT
                        *
               FD05     BSYHLT EQU     $FD05   *ｻﾌﾞBUSYﾌﾗｸﾞ,HALT Request
               FC80     SH.RAM EQU     $FC80   *ｷｮｳﾕｳ RAM ｾﾝﾄｳ Address
               9B50     CR.LF. EQU     $9B50   *ｷｬﾘｯｼﾞﾘﾀｰﾝ & ﾗｲﾝﾌｨｰﾄﾞ
               9BDB     OUTSTR EQU     $9BDB   *ﾓｼﾞﾚﾂ ﾉ ｼｭﾂﾘｮｸ
               AC3D     OUT_A  EQU     $AC3D   *A reg ｦ 16ｼﾝｽｳ ﾃﾞ ｼｭﾂﾘｮｸ
               DB59     BREAKC EQU     $DB59   *Breakｷｰ ﾁｪｯｸ ﾙｰﾁﾝ
                        *
                        *      MAIN Program
                        *
               4000     CLLIST EQU     *       *Start of MAIN Program
4000 0F        BF              CLR     $BF     *ｺﾝｿｰﾙ ｼｭﾂﾘｮｸ ｼﾃｲ
4002 86        07              LDA     #$07    *ｶﾗｰｺｰﾄﾞ ﾉ ｼｮｷｾｯﾃｲ
4004 A7        8D 00AE         STA     .COLOR,PCR *ｶﾗｰｺｰﾄﾞ ﾉ ｶｸﾉｳ
4008 B7        01E5            STA     $01E5   *ｶﾗｰｺｰﾄﾞ ﾉ ｶｸﾉｳ
400B 9E        35              LDX     $35     *Text End --> Xreg
400D 30        1E              LEAX    -2,X
400F 9F        A4              STX     $A4     *List Range End
4011 9E        33              LDX     $33     *Text Top --> Xreg
4013 9F        A2              STX     $A2     *List Range Top
4015 BD        9B50     CLLST0 JSR     CR.LF.  *CR & 'LF
4018 BD        DB59            JSR     BREAKC  *Break ﾁｪｯｸ
401B 9C        A4              CMPX    $A4     *List End ?
401D 25        03   4022       BLO     CLLST1  *No,then Continue.
                        *
401F 7E        8E72            JMP     $8E72   *ｺﾏﾝﾄﾞ ﾏﾁ ﾍ !!!
                        *
4022 9F        A2       CLLST1 STX     $A2
4024 EC        02              LDD     2,X     *ｷﾞｮｳﾊﾞﾝｺﾞｳ ﾉ ﾖﾐｺﾐ
4026 DD        A6              STD     $A6     *ﾋﾟﾘｵﾄﾞｷﾞｮｳﾊﾞﾝｺﾞｳ
4028 BD        B615            JSR     $B615   *ｷﾞｮｳﾊﾞﾝｺﾞｳ ﾉ ｼｭﾂﾘｮｸ
402B BD        9C22            JSR     $9C22   *ｸｳﾊｸ ﾉ ｼｭﾂﾘｮｸ
402E 9E        A2              LDX     $A2
4030 BD        C169            JSR     $C169   *1ｷﾞｮｳ ｷﾞｬｸﾎﾝﾔｸ ﾙｰﾁﾝ
4033 8E        043D            LDX     #$043D  *ﾊﾞｯﾌｧｾﾝﾄｳ --> Xreg
4036 9F        D9       CLLST2 STX     $D9     *ﾎﾟｲﾝﾀ ｾｰﾌﾞ
4038 C6        3A              LDB     #':     *ｸｷﾞﾘﾓｼﾞ ﾊ ｺﾛﾝ ($3A)
403A BD        90AA            JSR     $90AA   *ｸｷﾞﾘﾓｼﾞ ﾏﾃﾞ ｽｷｯﾌﾟ
403D E6        01              LDB     1,X     *ﾂｷﾞﾉｺｰﾄﾞ ﾉ ｾｰﾌﾞ
403F 34        14              PSHS    B,X     *ｾｰﾌﾞ
4041 6F        01              CLR     1,X     *Nullｺｰﾄﾞ ﾃﾞ ｸｷﾞﾙ
4043 9E        D9              LDX     $D9     *ﾓｼﾞﾚﾂｾﾝﾄｳ --> Xreg
4045 BD        D90F            JSR     $D90F   *1ｷﾞｮｳ ｼｭﾂﾘｮｸ ﾙｰﾁﾝ
4048 8D        0E   4058       BSR     CHGCOL  *ﾋｮｳｼﾞｼｮｸ ﾍﾝｶﾝ
404A 35        14              PULS    B,X
404C E7        01              STB     1,X     *ﾂｷﾞﾉｺｰﾄﾞ ﾉ ﾌｯｷ
404E A6        80              LDA     ,X+     *ｸｷﾞﾘﾓｼﾞﾊ.. NULL ? ｺﾛﾝ ?
4050 26        E4   4036       BNE     CLLST2  *ｺﾛﾝ ﾉ ﾊﾞｱｲ ﾊ ｷﾞｮｳ ｿﾉﾏﾏ
4052 9E        A2              LDX     $A2
4054 AE        84              LDX     ,X
4056 20        BD   4015       BRA     CLLST0  *ﾂｷﾞ ﾉ ｷﾞｮｳ ﾍ !!!
                        *
                        *      Subroutine
                        *
4058 A6        8D 005A  CHGCOL LDA     .COLOR,PCR *ｶﾗｰｺｰﾄﾞ ﾖﾐｺﾐ
405C 4C                        INCA            *ｺｳｼﾝ
405D 81        08              CMPA    #$08
405F 26        02   4063       BNE     CHGCL1
4061 86        01              LDA     #$01
4063 A7        8D 004F  CHGCL1 STA     .COLOR,PCR *ｶﾗｰｺｰﾄﾞ ﾉ ｶｸﾉｳ
4067 B7        01E5            STA     $01E5   *ｶﾗｰｺｰﾄﾞ ﾉ ｶｸﾉｳ
406A 30        8D 0035         LEAX    CLDATA,PCR * ﾃﾞｰﾀﾌﾞﾛｯｸ ｾﾝﾄｳ --> X reg
406E 108E      FC80            LDY     #SH.RAM * ｷｮｳﾕｳRAMｾﾝﾄｳ --> Y reg
4072 E6        8D 002C         LDB     LENGTH,PCR * ﾃﾝｿｳﾊﾞｲﾄｽｳ   --> B reg
4076 8D        03   407B       BSR     CM_TFR  * ｺﾏﾝﾄﾞ ﾉ ﾃﾝｿｳ
```

```
4078 8D   1D   4097          BSR    HRST      * HALT ﾉ ｶｲｼﾞｮ
407A 39                      RTS              *End of MAIN Program
                    *
407B 8D   08   4085 CM_TFR   BSR    HLT       *ｺﾏﾝﾄﾞﾉ ﾃﾝｿｳ
407D A6   80                 LDA    ,X+
407F A7   A0                 STA    ,Y+
4081 5A                      DECB
4082 26   F9   407D          BNE    *-5
4084 39                      RTS
                    *
4085 B6   FD05      HLT      LDA    $FD05     *HALT Request to ｻﾌﾞｼｽﾃﾑ
4088 2B   FB   4085          BMI    HLT       * if BUSY then Wait
408A 1A   50                 ORCC   #$50      * FIRQ,IRQ ｷﾝｼ
408C 86   80                 LDA    #$80      * BUSYﾌﾗｸﾞ ﾉ ｾｯﾄ
408E B7   FD05               STA    $FD05
4091 B6   FD05               LDA    $FD05     * Wait untill HALT ｱｸﾉﾘｯｼﾞ
4094 2A   FB   4091          BPL    *-3
4096 39                      RTS
                    *
4097 34   02        HRST     PSHS   A         *HALT ﾉ ｶｲｼﾞｮ
4099 4F                      CLRA
409A B7   FD05               STA    BSYHLT
409D 1C   AF                 ANDCC  #$AF      * FIRQ,IRQ ｷｮｶ
409F 35   02                 PULS   A
40A1 39                      RTS
                    *
40A2      14        LENGTH   FCB    L.DATA-CLDATA *ﾃﾞｰﾀ ﾉ ﾅｶﾞｻ
40A3      0000      CLDATA   FDB    $0000     *ﾃﾝｿｳﾃﾞｰﾀ
40A5      3F                 FCB    $3F       * TEST ｺﾏﾝﾄﾞｺｰﾄﾞ
40A6      59                 FCC    /YAMAUCHI/ * Key-word
40AE      91                 FCB    $91       * MOVE ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
40AF      D393               FDB    .COLOR-CLDATA+$D380
40B1      D02E               FDB    $D02E     * ｶﾚﾝﾄ ｱﾄﾘﾋﾞｭｰﾄ
40B3      0001               FDB    $0001     * ﾊﾞｲﾄｽｳ : 1
40B5      90                 FCB    $90       * END  ｻﾌﾞｺﾏﾝﾄﾞｺｰﾄﾞ
40B6      0001      .COLOR   RMB    1         * ｶﾗｰｺｰﾄﾞ ｶｸﾉｳﾊﾞｼｮ
          40B7      L.DATA   EQU    *         *End of DATA
                    *
          4000               END    CLLIST
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=4000
PROGRAM END   ADDR=40B6
PROGRAM ENTRY ADDR=4000
```

これに対して，＄Ｄ０Ａ０～＄Ｄ２ＢＦ番地の５４４バイトは，各処理系サブルーチン内において，上記のワークエリアを補足するための変数が配置されるエリアで，ローカル（局所的）変数にあたります．このエリアはすべての処理系で使用されるわけではありませんし，使われ方は各処理系によってまちまちです．ＰＡＩＮＴ処理系などでは,このエリアを第2スタックとして使用しております．

上記２つのワークエリアについての詳細な内容は，第５章「ワークエリア対照表」および第３章の各節に掲載してありますので随時参照して下さい．

＄Ｄ２Ｃ０～＄Ｄ３６０の１６０バイトは，ＰＦキー定義テーブルです．このテーブルの構造は，図２・２・２のようになっており，各ＰＦキーに対してそれぞれ１６バイトずつのテーブルが割り当てられます．各テーブルの先頭の１バイトは，割込み設定状態とＰＦキーに定義されている文字列の大きさ（１～１５バイト）を表すフラグレジスタになっており，残りの１５バイトに実際の文字列のコードが格納されます.フラグレジスタの最上位ビットで割込み設定状態を表し，該当するＰＦキーに対して割込みが設定されている時は，このビットが“１”に

なります．下位４ビットで文字列の大きさ（バイト数）を表します．

＄Ｄ３６０～＄Ｄ３７Ｆの３２バイトはキー入力バッファです．ＦＭ－７ではバッファモード（キー先行入力許可状態）というモードがあり，キー入力バッファの果たす役割は，より一層大きくなります．

＄Ｄ３８０～＄Ｄ３ＦＦ番地の１２８バイトは，第１章でも述べたように共有ＲＡＭの領域です．

＄Ｄ４００～Ｄ７ＦＦ番地は，Ｉ／Ｏポートのための領域です．いわゆる，「Ｉ／Ｏ領域」として割り付けられている空間は，１Ｋバイトにもなりますが，実際に使用されている空間は，＄Ｄ４００～Ｄ４０Ｆ番地までの１６バイトに過ぎず，残りの空間は，アドレスがフルデコードされていないために眠っています．ワークエリアＲＡＭ，共有ＲＡＭ用として装着されている２ＫバイトＲＡＭ：ＭＢ８１２８（Ｍ１２２）の後ろ半分が全く使用されないまま余っていること，サブシステム内において自由な目的に使用できるＲＡＭがほとんど無いことを考えると，この部分がフルデコードされていてＲＡＭにつながれてさえいれば，あと１Ｋバイト近くのＲＡＭを自由に使うことができるのに‼と残念でなりません．ハードウェアに関して腕に自信があり，ＦＭ－７の改造を企ててやろうという勇敢な方は，この部分をフルデコード化して，余った空間に対して先のＲＡＭをつなぐとよいでしょう．ＴＴＬのＩＣを３つばかり補うだけでできると思います．ただし，積層基板を敵にまわしての改造は，相当な困難を伴います．よほどの覚悟がないとできないことでしょう．

さてサブシステムのＩ／Ｏポートは，図２・２・３に示すとおり全部で11個です．＄Ｄ４０３（ＢＥＥＰ音発生）と＄Ｄ４０Ｄ（ＩＮＳモード表示ＬＥＤ点滅）の２つを除けば，どれも重要かつ深遠な意味を持っており，曲者といってよいでしょう．

＄Ｄ８００～＄ＤＦＦＦ番地の４Ｋバイトは，キャラクタフォントの内容が格納されているＲＯＭに割り当てられます．キャラクタジェネレータＲＯＭに相当するものですが，ＦＭ－７では表示文字コード→表示文字パターンの変換を，ハードではなくソフト的に行っているため，メモリ空間上にこのようなＲＯＭが必要になっているのです．

＄Ｅ０００～＄ＦＦＦＦ番地の８Ｋバイトは，第３章以下に解説しますが，サブシステムモニタ（制御用プログラム）の全内容が収納されているＲＯＭの空間です．いわばサブシステムの頭脳といってよいでしょう．

以上のメモリマップは，ＦＭ－８では若干の相違が見られますので注意して下さい．

図2・2・2　ＰＦキー定義テーブルの様子

	+0 b7　　　　b3 ～ b1		+1 ………………………… +15
＄Ｄ２Ｃ０	IF_1　(未使用)	ＰＦ１キーの文字列サイズ	ＰＦ１キーの文字列の内容
＄Ｄ２Ｄ０	IF_2　(　〃　)	ＰＦ２キーの文字列サイズ	ＰＦ２キーの文字列の内容
＄Ｄ２Ｅ０	IF_3　(　〃　)	ＰＦ３キーの文字列サイズ	ＰＦ３キーの文字列の内容
	⋮	⋮	⋮
＄Ｄ３５０	IF_{10}　(　〃　)	ＰＦ10キーの文字列サイズ	ＰＦ10キーの文字列の内容

※IF_1～IF_{10}：　各ＰＦキーに対して，ＫＥＹ割込みが設定されているか否かを表すフラグ（割込みが設定されている時このビットは"１"になる）

先程も少しふれましたが，サブシステムのメモリマップを見てまず目立つことは，ユーザが自由な目的に使うことの可能な，完全に開放されているＲＡＭエリアというものがほとんどないということです．なんといってもサブシステムを使いこなしていく上で，最大の障害となるのはこのことであり，いかにＲＡＭエリアを確保するかが，重要な鍵となることでしょう．

では，どのようにＲＡＭエリアを確保すればよいのでしょうか．

短かいプログラムやデータであれば，共有ＲＡＭ内に置いて走らせることができます．またスタックをほとんど使わないようなプログラムなら，スタックエリアの前の方（＄ＣＦＡ１～）を使用することができます．いずれにせよ容量を大きく取ることはできません．

場合に応じて補助ワークエリア（＄Ｄ０Ａ０～＄Ｄ２ＢＦ）を使うのは良い方法です．このエリアは通常，ほとんど全体が開放されて使用可能な状態となっているからです．ただし，ＧＥＴ，ＰＵＴ，ＳＹＭＢＯＬ…などの処理系では前の方の一部分が，ＰＡＩＮＴなどの処理系では第２スタックとしてほぼ全体が使用されるため，ユーザがこれらの処理系およびその内部サブルーチンと呼び出して使用する際には，細心の注意を払う必要があります．この方法では最高５４４バイトのＲＡＭエリアを確保することができます．

上記の方法でも収まらない場合，例えばサブシステム内にゲームを収めようとか，グラフィックツールのサブルーチンをロードしようなどといった場合には，通常の文字入出力（スクリーンエディタ，ＰＵＴコマンド等）を諦らめ，コンソ

ールバッファ（＄Ｃ０００～＄ＣＦＡ０）の約４キロバイトエリアをつぶして使うより他ありません．この際，コンソールバッファの内容を，メイン側の裏ＲＡＭなどに退避しておくと，復帰させるのが容易です．

図２・２・３　サブシステムのＩ/Ｏポートのアドレス割付け

アドレス	機　　能	Read/Write	Reset 時の設定	補　足　説　明
＄Ｄ４００	キーボードデータ（補助ビット）	Read オンリー	――	キー入力がＰＦキーの場合，最上位ビットが〝１〟他は〝０〟
＄Ｄ４０１	キーボードデータ（D_0～D_7）	Read オンリー	――	キーボードエンコーダからの８ビットデータ（キーコード）
＄Ｄ４０２	キャンセル割込みアクノリッジ	アクセス（Ｒ／Ｗ両方可）	割込みクリア	メインからのキャンセル割込み信号をクリアする．
＄Ｄ４０３	ブザー（ＢＥＥＰ音発生）	アクセス	――	
＄Ｄ４０４	アテンション割込み発生	アクセス	割込みクリア	アテンション割込みをメイン側に要求する．
＄Ｄ４０５	《未使用》	――	――	
＄Ｄ４０６	《未使用》	――	――	
＄Ｄ４０７	《未使用》	――	――	
＄Ｄ４０８	ＣＲＴ表示ON/OFF	Ｒ：表示ＯＮ Ｗ：表示ＯＦＦ	表示ＯＦＦ	ＣＲＴ表示のＯＮ／ＯＦＦを制御する．
＄Ｄ４０９	ＶＲＡＭアクセスフラグ　※	Ｒ：Set Ｗ：Reset	Reset	サブＣＰＵがＶＲＡＭアクセス中であることをＣＲＴＣに知らせる．
＄Ｄ４０Ａ	ＢＵＳＹフラグ	Ｒ：Ready Ｗ：ＢＵＳＹ	ＢＵＳＹ	サブＣＰＵがコマンド待ち状態であることをメイン側に知らせる．
＄Ｄ４０Ｂ	《未使用》	――	――	
＄Ｄ４０Ｃ	《未使用》	――	――	
＄Ｄ４０Ｄ	ＩＮＳモード表示用ＬＥＤ点滅	Ｒ：点灯 Ｗ：消灯	消　灯	ＩＮＳ（インサート）モード状態表示用ＬＥＤの点灯/消灯を制御する．
＄Ｄ４０Ｅ	ＶＲＡＭオフセットレジスタ（Ｈ）	Write オンリー（Read 禁止）	＄００	ＣＲＴＣ内のレジスタで，ＶＲＡＭのハードウェア的スクロールを制御する（Read すると値は不定になるので，Read 禁止）．
＄Ｄ４０Ｆ	ＶＲＡＭオフセットレジスタ（Ｌ）	Write オンリー（Read 禁止）	＄００	

※ＶＲＡＭアクセスフラグを立てた直後にＶＲＡＭをアクセスする場合には，間に１マシンサイクル以上のディレイを挿入する必要があります。

2－3　メイン↔サブ間インタフェース

まず，図２・３・１の概念図および図２・３・２のブロック図を見て下さい．

メイン↔サブ間インタフェースの主役は，なんといっても共有ＲＡＭです．共有ＲＡＭというのは，メイン↔サブ間のコミュニケーションを図るための重要なウィンドウエリアで，通常はサブ側に接続されていますが，サブＣＰＵをＨＡＬＴ（停止）させることにより，メイン側に接続することができ，これによってメインＣＰＵは，サブシステム内の情報を読み取ったり，反対にサブシステムに情報を送り込んだりすることが可能なわけです．

それでは，各信号線ならびにレジスタの意味を見ていくことにします．

(1)ＨＡＬＴ

サブシステムの動作を停止させ，メイン側からの共有ＲＡＭ参照を可能にするための信号線です．なお図２・３・２の②に示すように，サブＣＰＵに対してＨＡＬＴがかかるのは，メイン側からの要請があった時だけではなく，ＣＲＴＣ（ＣＲＴコントローラ）がＶＲＡＭスキャンを行う際にも生じる可能性があります．これはサブＣＰＵがＶＲＡＭアクセスフラグを立ててＶＲＡＭをアクセスする場合に，ＣＲＴＣとの競合を防ぐためのものです．従ってＶＲＡＭアクセスフラグを立てた場合，サブＣＰＵがアクティブ（動作状態）であるのは，水平および垂直帰線期間中だけに限られてしまい，サブＣＰＵの稼動率は３２％にまで低下します．先に述べたメモリ空間の制約とともに，サブＣＰＵは速度の点でも限界があることをよく認識しておいてください．サブシステムの各コマンド処理系のアルゴリズムは，サブＣＰＵの性能をほとんどフルに発揮しているものと思われ，ユーザがサブシステムを使いこなすことによって処理の高速化を図る余地は，ほとんどないものと見受けられます．他メーカーのパーソナルコンピュータのように，画面がちらついても構わないからその分高速化を図る，という動作をさせるのは，ＦＭ－７/８の場合にはハードウェアの改造を行わない限り不可能です．

ただし，以上の話はＦＭ－７７では様相が違ってきます．ＦＭ－７７では，ＣＲＴＣによるＶＲＡＭ制御にサイクルスチールという技法を導入したので，競合を防ぐための時間待ち用ＨＡＬＴは不要になっております（完全互換モードでは時間待ちＨＡＬＴが残されています）．ＶＲＡＭ制御におけるサイクルスチールは，次のように行われているものと思われます．

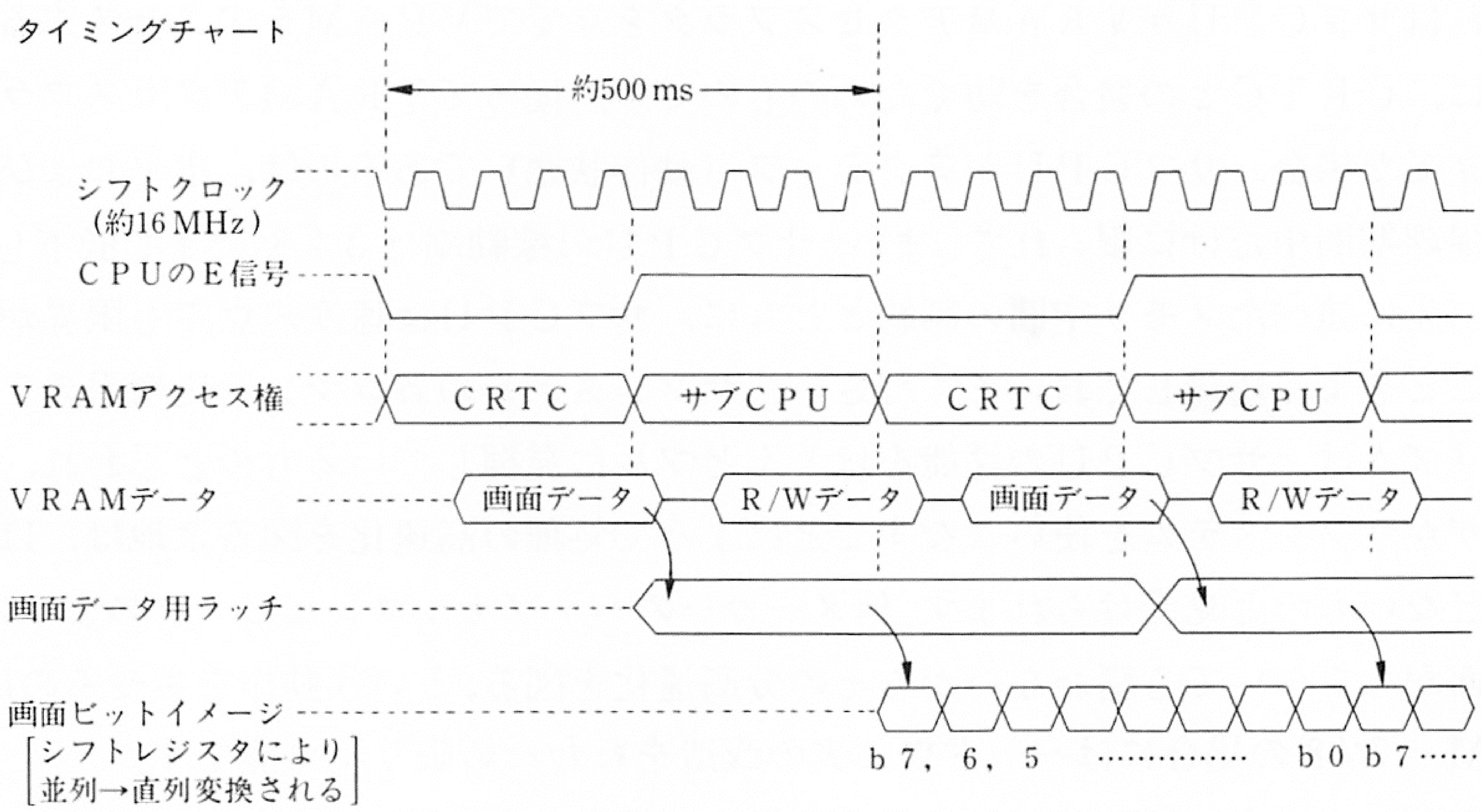
まず，ＣＰＵに外部クロック同期タイプである６８Ｂ０９Ｅを使用することによって，２ MHz（正確には２．０１６ MHz）クロックでＣＰＵとＣＲＴＣを完全に同期させます．すると，ＣＰＵに加わるＥ信号がＬレベルの期間中は，ＣＰＵ

から見ると完全にバスが空いた状態になります．ですからこの期間をＣＲＴＣによるＶＲＡＭアクセスのための時間に割り当てるわけです．つまり，ＣＰＵの１サイクルのうちの前半ではＣＲＴＣが，後半ではＣＰＵが，それぞれＶＲＡＭアクセス権を持つようにするわけです．これを実現させるためには，メモリのサイクルタイムが250ms以下でなければならず，高速ＲＡＭが実装されております．この技法を用いることによって，ＣＰＵとＣＲＴＣの競合を防ぐための時間待ちＨＡＬＴという余計な機構は不必要となり，しかも，サブシステムのソフトウェア自体には全く変更を加えずに，スループットを高めて処理の高速化を実現することができるわけです．

この「ソフトウェアの変更が全く不要」という点は非常に重要で，ハードウェア設計の際に参考にして下さい．

完全互換モードではＨＡＬＴがかかる一方，ＦＭ－７７独自の高速モードではＶＲＡＭアクセスフラグのアクセスは，ダミー的なものとなります．

それでは，筆者が推察したタイミングチャートを下図にまとめます．



(2)ＢＵＳＹフラグ

サブシステムがコマンド実行中などで，ＨＡＬＴをかけて停止させると不都合な状態であることを，メイン側に伝えるための信号線です．サブシステムのＩ／Ｏポート＄Ｄ４０Ａのリード（Ready）／ライト（Busy）で制御します．ただし，メイン側からのＨＡＬＴ要求を受け付けた場合，ＢＵＳＹフラグは自動的に（つまりハードウェア的に）Ｈレベル（Busy状態）に戻るような構造になっていて，ＨＡＬＴアクノリッジ信号の役目をも果たしております．

図2・3・1　メイン↔サブ間インタフェース回路の概念図

①　共有ＲＡＭと制御用信号線



②　共有ＲＡＭの標準的構成

サブ			メイン
$D380	Ready Request(ビット7)	エラーコードレジスタ(ビット6～0)	$FC80
1	CONTフラグ(ビット7)	ステータスレジスタ　(ビット6～0)	1
2	コマンドコードレジスタ($01～0D，15～20，……，64)		2
3	パラメータ等のデータ		3
$D3FF			$FCFF

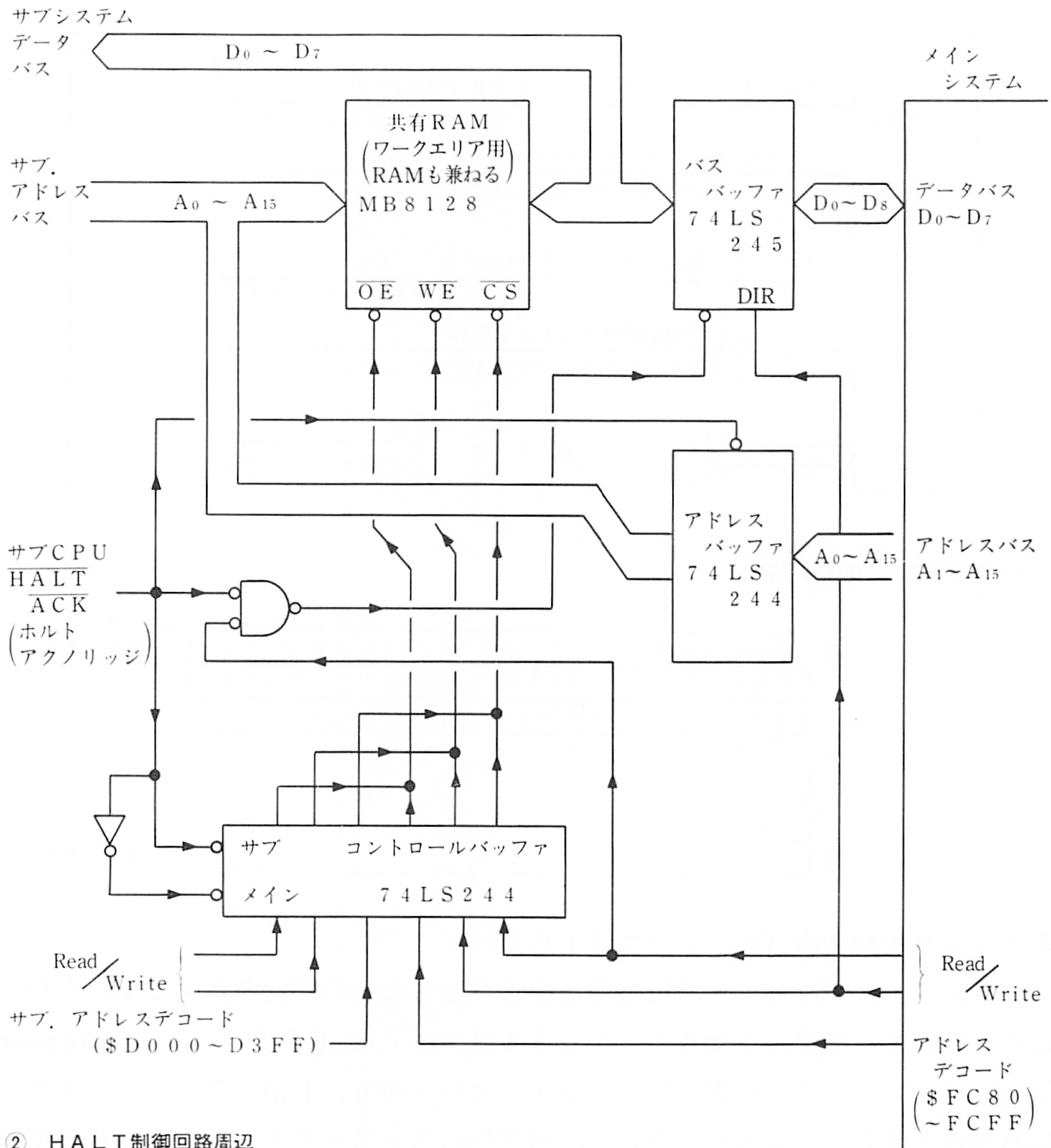
(3)キャンセル割込み（メイン→サブＩＲＱ）

サブシステムのコマンド実行処理を中止させるための割込み信号です．ただしこの信号によって中止させることができるコマンドは，ＧＥＴコマンドと，Graphic Cursorコマンドの２つに限られています．というのも，FM－7のサブシステムにおけるＩＲＱサービスルーチンは，以下のリストに示すように，ＡＢＯＲＴフラグ（$Ｄ０００）を立てて戻るだけであるため，このＡＢＯＲＴフラグをチェックしている上記の処理系でなければ，動作を中止させることができないからです．この他ＰＵＴコマンド処理系において，ＥＳＣキーによるプットウェイト状態を解除するための要因として，例外的に，ＡＢＯＲＴフラグがチェックされます．

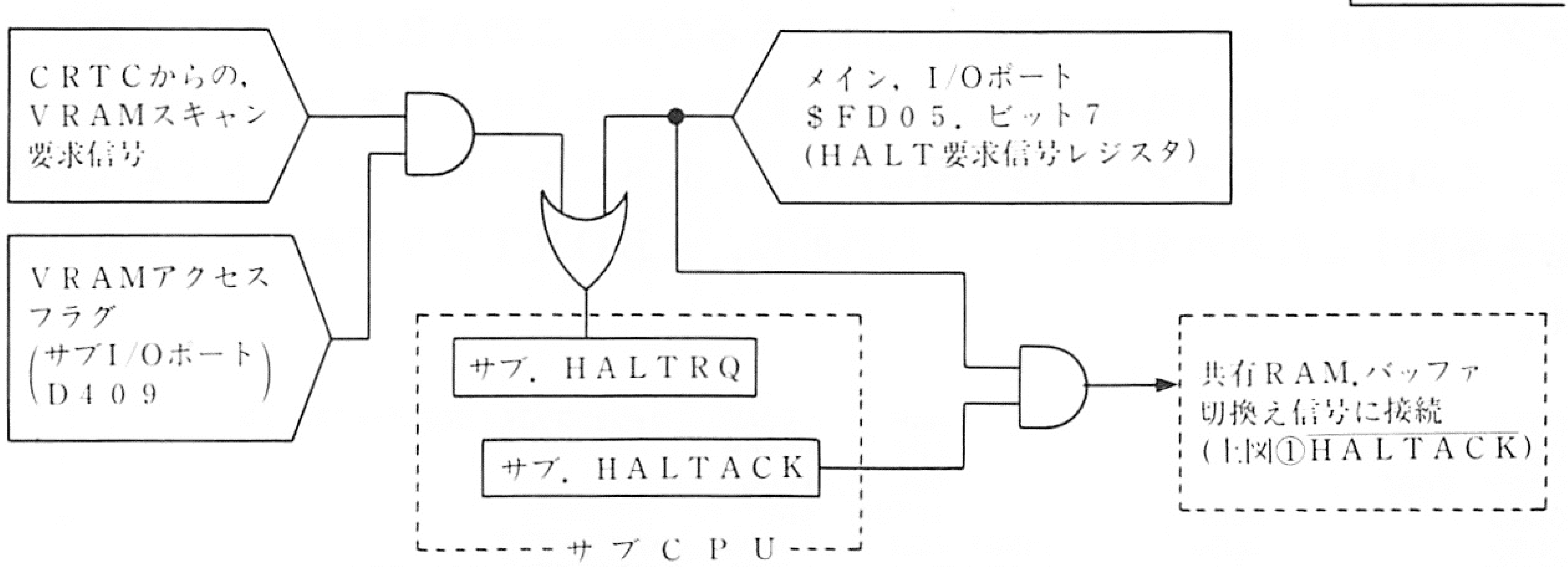
```
        E06E                       NAM     IRQAB     ｷｬﾝｾﾙﾜﾘｺﾐ (IRQ) ｻｰﾋﾞｽﾙｰﾁﾝ
                                   OPT     NOMEM
        E06E                       ORG     $E06E
                   00D0            SETDP   $D0
                   D000    ..ABRT  EQU     $D000     *ｱﾎﾞｰﾄﾌﾗｸﾞ
                   D402    ACKIRQ  EQU     $D402     *IRQ ｱｸﾉﾘｯｼﾞｼﾝｺﾞｳ
```

図2・3・2　メイン↔サブ間インタフェース回路ブロック図

① 共有ＲＡＭ周辺



② ＨＡＬＴ制御回路周辺



```
                                *
             E06E       IRQAB  EQU    *
E06E B5      D402              BITA   ACKIRQ   *IRQ ｱｸﾉﾘｯｼﾞ
E071 86      FF                LDA    #$FF     *Set Abort Flag
E073 97      00                STA    ..ABRT   *ｱﾎﾞｰﾄﾌﾗｸﾞ ON
E075 3B                        RTI             *ﾓﾄﾉｼｮﾘｹｲ ﾆ ﾘﾀｰﾝ
                                *
                               END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=E06E
PROGRAM END   ADDR=E075
PROGRAM ENTRY ADDR=****
```

なお，ＦＭ－８の場合には，キャンセル割込みの様子が異っております．

キャンセル割込みを受け付けることが可能な処理系では，＄Ｄ０００～＄Ｄ００１番地の２バイトに，処理系ごとのサービスルーチンへのジャンプベクトルを書き込んでおいてからＩＲＱ許可状態にしておき，ＩＲＱがかかった際には，そのベクトルで示される各サービスルーチンを呼び出すようになっています．通常は，ベクトルとして＄００００（ＮＵＬＬマーク）が書かれており，ＩＲＱ禁止の状態になっていますが，ＰＵＴ処理系におけるプットウェイト状態の場合などでは，ベクトルは＄００００のままＩＲＱ許可状態になります．この状態でＩＲＱが受け付けられた場合には，即座にコマンド待ちルーチン（メインループ）に飛ぶようになっています．

いずれにしても，ＩＲＱサービスルーチンのシンプルさは，ユーザにとっては，割込みを利用した様々なアプリケーションの余地を残すものであり，キャンセル割込みをうまく活用することによってメイン↔サブ間の同期をとるなど，なかなか面白いことができます（１－２－３節のサンプル――例１・２・５参照）．

なお，メイン側からのキャンセル割込みのかけ方ですが，Ｉ／Ｏポート＄ＦＤ０５のビット６を"１"にして割込み要求を行った後，すぐにまた，"０"に戻しておく必要があります．"１"にしたままほうっておきますと，次の割込みがかかりません．

⑷アテンション割込み（サブ→メインＦＩＲＱ）

メインシステムからあらかじめ依頼されていた割込み事象が発生したことを，サブ側からメイン側に通知するための割込み信号です．該当する割込み事象は，ＰＦキー割込み・タイマ割込み・インタバル割込み・０時割込みの４つです．

このアテンション割込みもキャンセル割込みと同様，うまく活用することによって，様々な面白いアプリケーションが可能な信号線です．

⑸共有ＲＡＭ

第１章でふれたとおり，メイン↔サブ間においてコマンドおよびパラメータなどの情報交換に用いられるＲＡＭです．標準的な構成は図２・３・１に示すとお

りです．それでは各バイトの内容について解説していきます．

① Ready Request フラグ

共有ＲＡＭの先頭バイトの最上位ビット（サブ＄Ｄ３８０ビット７）で，サブシステムにＨＡＬＴをかけて新たにコマンドを設定しない場合，例えば，サブシステムに実行させたコマンドの結果をメイン側から読み込むケースなどは，このビットを“１”にセットします．というのは，サブシステムにＨＡＬＴをかけた場合，ＢＵＳＹフラグが自動的に（つまり，ハードウェア的に）Ｈレベル（Busy 状態）に戻るので，そのままにしておくとメイン↔サブの通信に不都合が生じます．そこで，ＢＵＳＹフラグをＬレベルに戻して，再びコマンド待ち状態（Ready 状態）に入るように，サブシステムに指示する必要があるからです．ただし，コマンドコードレジスタに何か値を入れてＨＡＬＴを解除した場合には，コマンドの実行が優先され，この Ready Request フラグは無視されます．

② エラーコードレジスタ

共有ＲＡＭ先頭（＄Ｄ３８０）の下位７ビット（ビット０～６）には，コマンド実行時，コマンドもしくはパラメータに誤りがあった場合に，どのような誤りであるかを表すエラーコードが設定されます．エラーコードとエラー内容の対応は次表のとおりです．

エラーコード	エ　ラ　ー　内　容
＄３Ｃ(６０)₁₀	ＩＮＩＴコマンドのパラメータエラー※
＄３Ｄ(６１)	コンソール座標エラー
＄３Ｅ(６２)	オーダのオペランドエラー
＄３Ｆ(６３)	グラフィック座標エラー
＄４０(６４)	ファンクションコードエラー
＄４１(６５)	座標数エラー
＄４２(６６)	文字数エラー
＄４３(６７)	色指定数のエラー
＄４４(６８)	ＰＦキー番号エラー
＄４５(６９)	その他のパラメータエラー
＄４６(７０)	コマンドコードエラー

※ＩＮＩＴコマンドのパラメータエラーが発生した場合には，暗黙設定（デフォルトイニシァライズ）がなされるので注意が必要です．

なお，ＢＩＯＳ経由でサブシステムにコマンドを送出した場合には，これら

のエラーコードがそのままＢＩＯＳのエラーコードとなります．

③　ＣＯＮＴフラグ（データ継続フラグ）

共有ＲＡＭ第２バイト（＄Ｄ３８１）の最上位ビットは,，サブからメインへの送出データ，あるいはメインからサブへの入力データが１回の転送では終了しない場合に，"１"にセットして，データが継続していることを知らせるためのフラグとして用いられています．サブからメインへとデータが送られている場合にはサブ側がセットして，それをメイン側で参照し，それに応じた処理をすることになりますが，メインからサブへとデータを転送する際には，ＣＯＮＴフラグはメイン側でセットしてやる必要があります．ＣＯＮＴフラグが"１"となるケースで，継続データの入出力を要求する場合には，専用のコマンドであるＣＯＮＴ（コマンドコード：＄６４）を使用します．

④　ステータスレジスタ

＄Ｄ３８１の下位７ビットは，アテンション割込みの際に，割込みの要因を表すためのステータスレジスタとして使用されます．メイン側では，アテンション割込み用のサービスルーチンにおいて，このレジスタを読み込むことによりどのような割込み事象が発生したかを判別し，割込み事象に応じた処理を行う必要があります．レジスタ内の各ビットと割込み事象との対応は次表のとおりです．

ビットNo.	割込みの内容
ビット０～３	ＰＦキー割込みのキーコード(１～１０)※
ビット４	タイマ割込み(割込みが生じた場合に"１"が入る)
ビット５	インターバル割込み(　同上　)
ビット６	０時割込み　(　同上　)
(ビット７)	(前述のＣＯＮＴフラグ)

※ＰＦキーによる割込みが生じていない場合には，ビット０～３には＄０が入る．

⑤　コマンドコードレジスタ

＄Ｄ３８２（メイン＄ＦＣ８２）は，メインからサブへと要請されるコマンドのコードを入れるためのレジスタです．

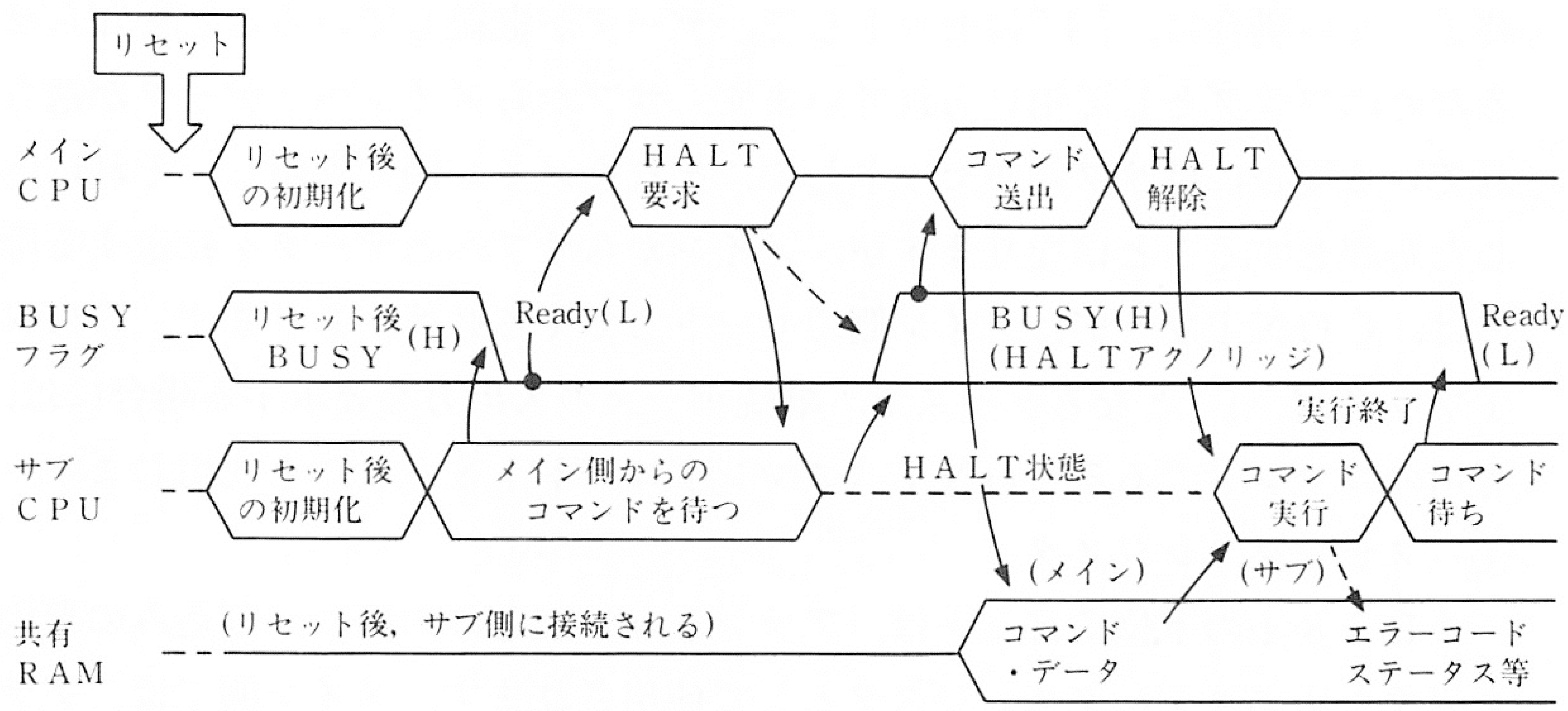
各コマンドに必要なパラメータ類は，共有ＲＡＭの４バイト目（サブ：＄Ｄ３８３，メイン：＄ＦＣ８３）以降に配置されています．

それでは，以上に述べてきたメイン↔サブ間インタフェースの様子を，４つの

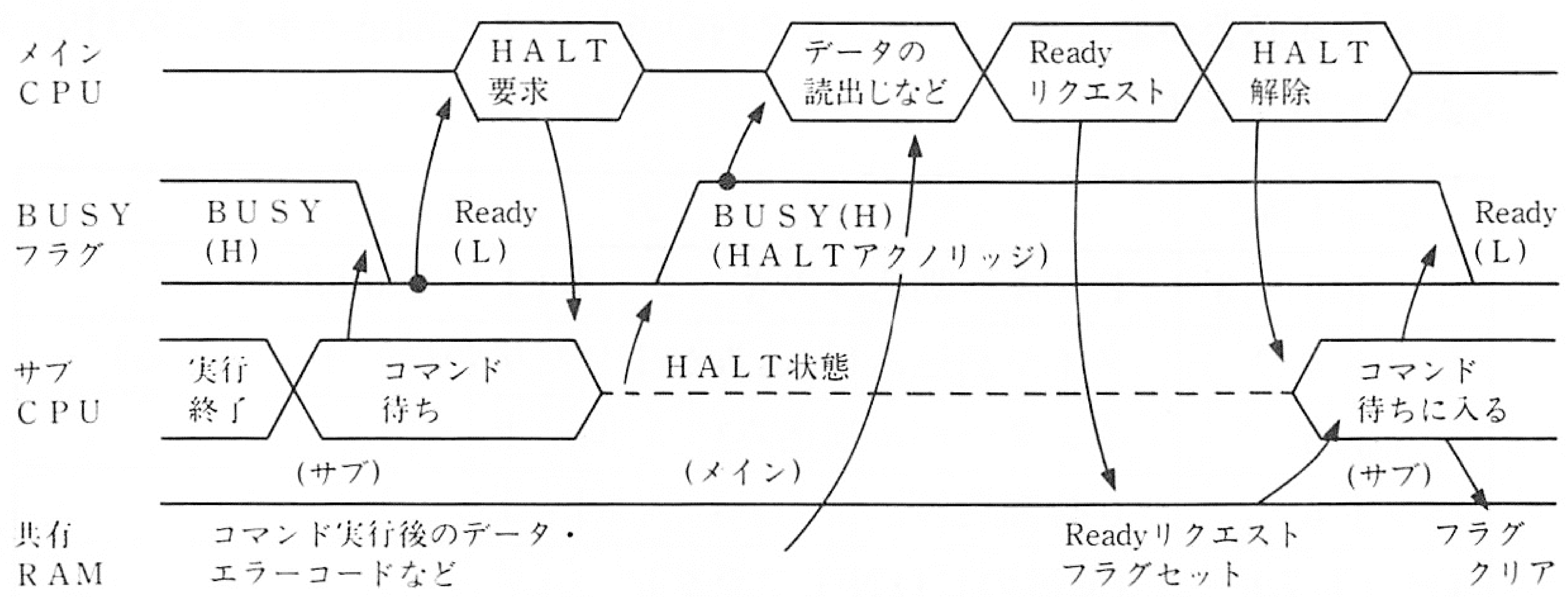
パターンに分けて図解します.

図2・3・3　メイン↔サブ間インタフェースのタイミングチャート(2の1)

パターン1　リセットからコマンドの送出まで



パターン2　Readyリクエスト(データ読出しなどのケース)

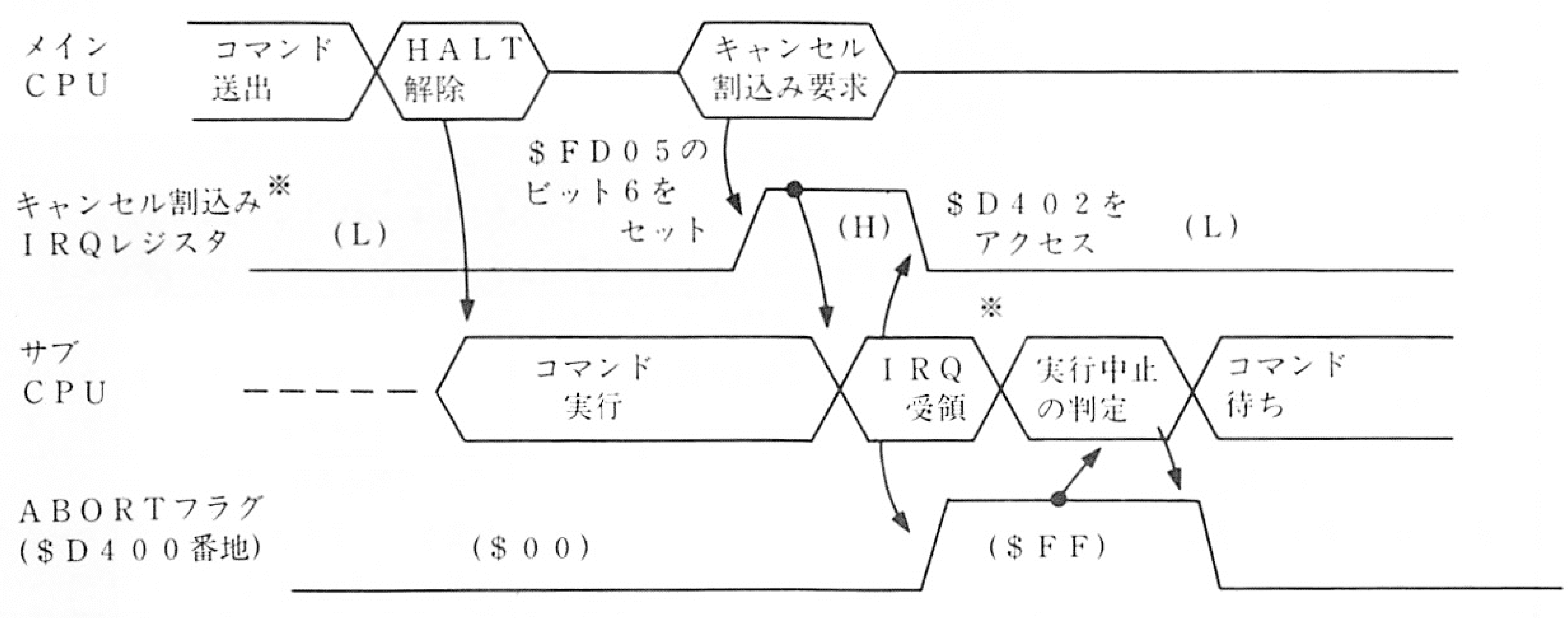


最後に，サブシステムと関係が深いメイン側Ｉ／Ｏポートについて46ページにまとめておきます.

ここで特に気をつけなくてはならないのは，＄ＦＤ０２番地ビット０に位置するキーボード割込みのマスクについてです．このビットは，キーボードからの割込み信号が加わる先を，サブ側（ＦＩＲＱ）にするか，メイン側（ＩＲＱ）にするかを決定する切換えスイッチで，通常は，割込み信号がサブ側に加わるように設定されており，このビットの値によって，メイン側に接続されるようにできます．キーボードデータは，常にどちらのＣＰＵからでも読込み可能になっており

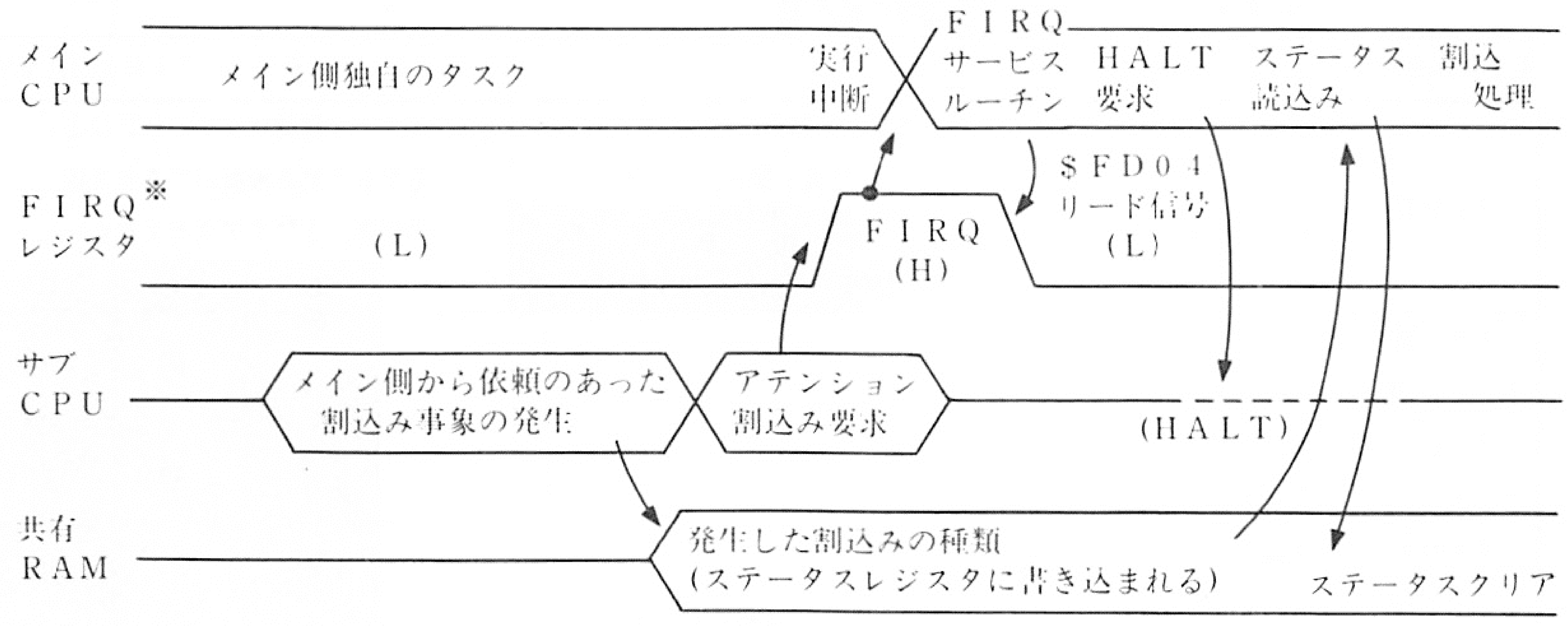
図2・3・3　メイン↔サブ間インタフェースのタイミングチャート(2の2)

パターン3　キャンセル割込み(メイン→サブIRQ，コマンド実行中止の要求)



※キャンセル割込みIRQレジスタは，メイン側$FD05番地ビット6への"1"の書込みによりセット，サブ側$D402番地のアクセスによりクリアされるようになってます．もちろん，リセットがかかった場合には自動的にクリアされます．
※キャンセル割込みIRQレジスタのクリア，ABORTフラグのセットは，サブ側IRQサービスルーチンによって行われます．図中「IRQ受領」と書いてある部分がサービスルーチンに相当します．

パターン4　アテンション割込み(サブ→メインFIRQ，要請された割込み事象発生の通知)



※FIRQレジスタは，サブ側$D404番地のアクセス，Breakキー押下によりセットされ，メイン側$FD04番地(FIRQ要求元)の読込みによりクリアされるようになっています．もちろん，リセットがかかった場合には自動的にクリアされます．

★灰色の部分がサブシステムと関係の深いエリアです．

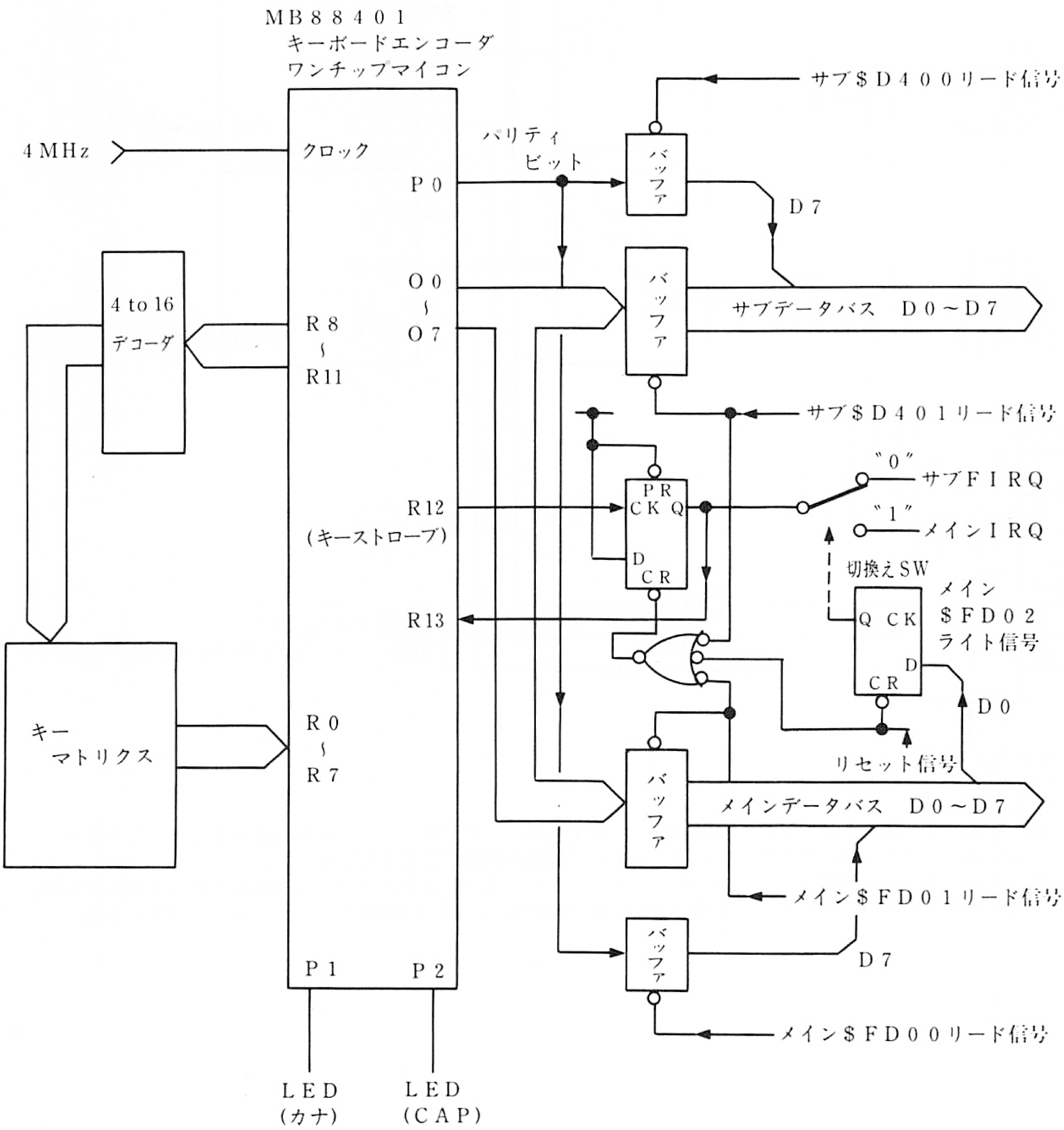
アドレス	内容	リード ライト	7	6	5	4	3	2	1	0
$FD00	キーデータ（パリティビット）クロック状態	リード	PFキーのときに"1"							2MHzモードで"1"
$FD01	キーデータ（ビット0～7）	リード	キーボードデータ PFキー（パリティ＝"1"）のときPFキー番号（ビット0～3） 通常キー（パリティ＝"0"）のとき，Asciiコード（ビット0～7）							
※ $FD02	IRQ割込みのマスクレジスタ	ライト	"0"で割込み禁止／"1"で割込み許可 （RS－232C関係の割込み）					タイマ（メイン）	プリンタ	キーボード
$FD03	IRQ割込みの要求元を表すレジスタ	リード					"0"で割込み有り!! 拡張カードから．	タイマ（2mS）	プリンタ	キーボード
$FD04	FIRQ割込みの要求元を表すポート	リード							Breakキー押下状態で"0"	アテンション有りで"0"
$FD05	メイン↔サブ間インタフェース信号	リード	BUSYフラグ（$B/\overline{R}$）							$\overline{\text{EXT}}$ $\overline{\text{DET}}$（※）
	メイン↔サブ間インタフェース，Z80切換えSW	ライト	HALT要求	キャンセル割込要求（"1"でアクティブ）						Z80W（※）

※$FD02の内容は，リセット時に"$00"となります．　（※フェーズIおよびII参照）

アドレス	内容	リード ライト	7	6	5	4	3	2	1	0
$FD37	マルチページ制御レジスタ	ライト	ディスプレイ制御（"1"で表示禁止）	G（グリーン）	R（レッド）	B（ブルー）	アクセス制御（サブCPUによるVRAM各画面のアクセス権の制御，"1"でアクセス禁止）	G	R	B
$FD38 コード0（黒）	パレットレジスタ ※VRAMを指定する際の論理的カラーコードと，実際にCRTに表示されるカラーコード（物理的カラーコード）との対応づけを行うためのレジスタ	R/W	実際にCRT上に表示される色のカラーコード（物理的カラーコード）					G	R	B
$FD39 コード1（青）		R/W						G	R	B
$FD3A コード2（赤）		R/W						G	R	B
$FD3B コード3（紫）		R/W						G	R	B
$FD3C コード4（緑）		R/W						G	R	B
$FD3D コード5（水色）		R/W						G	R	B
$FD3E コード6（黄）		R/W						G	R	B
$FD3F コード7（白）		R/W						G	R	B

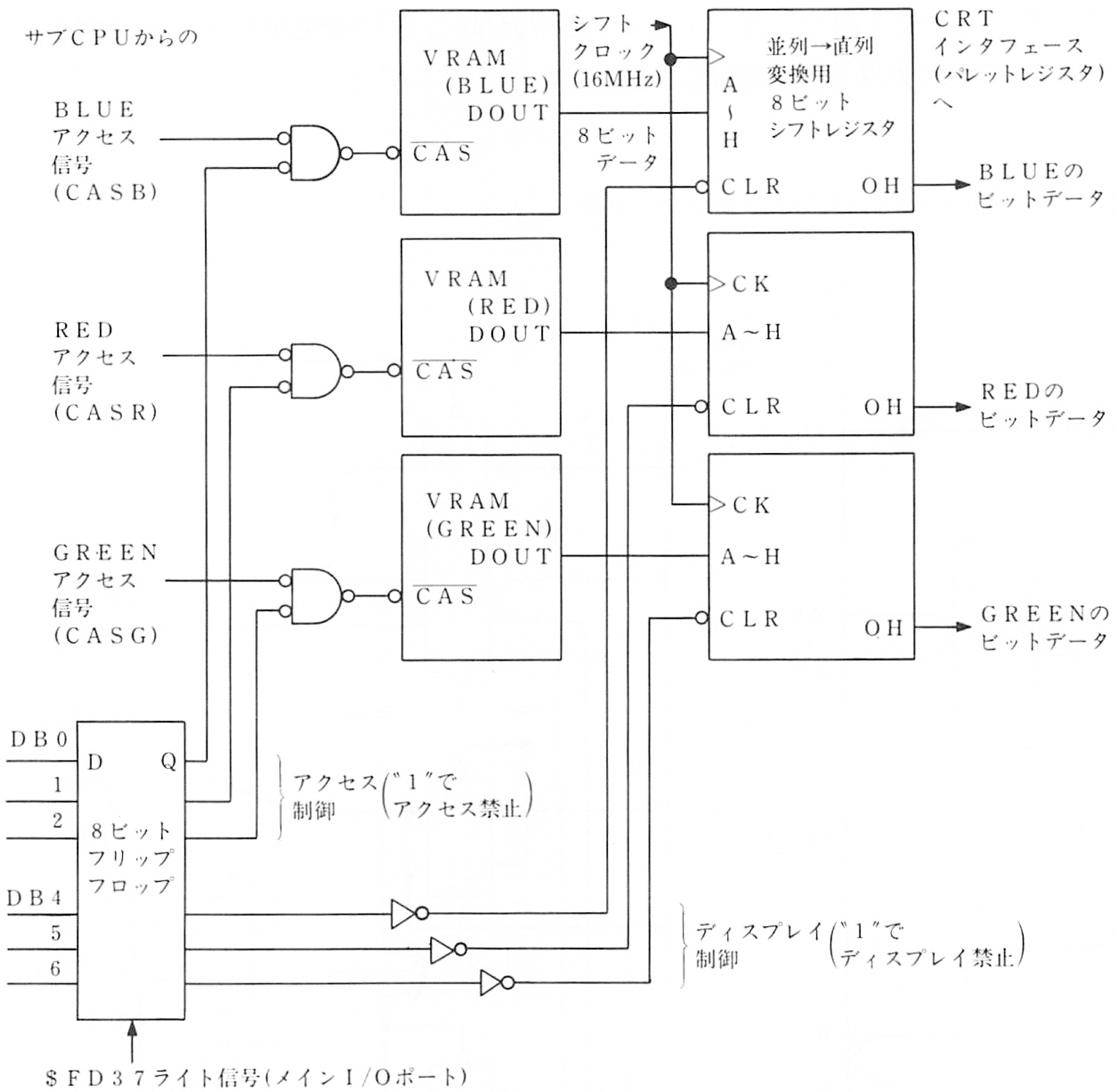
ますが，もし割込み信号の接続されていない方のＣＰＵが，キーボードデータを読み込んでしまう（メインが＄ＦＤ０１，サブが＄Ｄ４０１をリードする）と，その時点で自動的にキーボード割込み用のフリップフロップ（キーストローブ）をクリアしてしまうので，キーデータの消失が起こる可能性があります（図２・３・４）．くれぐれも誤った読出しをしないようにして下さい．

図2・3・4　キーボードインタフェース部のブロック図



一方，ＶＲＡＭのマルチページ制御は，図２・３・５のように行われます．

図２・３・５　マルチページ制御の様子(この図では，ＣＲＴＣからの信号は省略してあります)



※　アクセスが禁止されたＶＲＡＭ領域をリードすると，"＄ＦＦ"のデータを読み取ることになります．これは，データバスに付加されているプルアップ抵抗の働きによるものです．

※　また，アクセスが禁止されていても，オフセットレジスタによるスクロールは通常どおり行われますが，スクロールによってせり上がる領域のクリアが行われないため，画面が見づらくなることがあります．

2－4　ＣＲＴＣによるＶＲＡＭ制御

ＶＲＡＭ部分のブロック図２・４・１に掲げます．このうちユーザにとって最も関心のある部分は，ＶＲＡＭのアクセス権の問題と，オフセットレジスタの書換えによる高速スクロールの模様の２点でしょう．

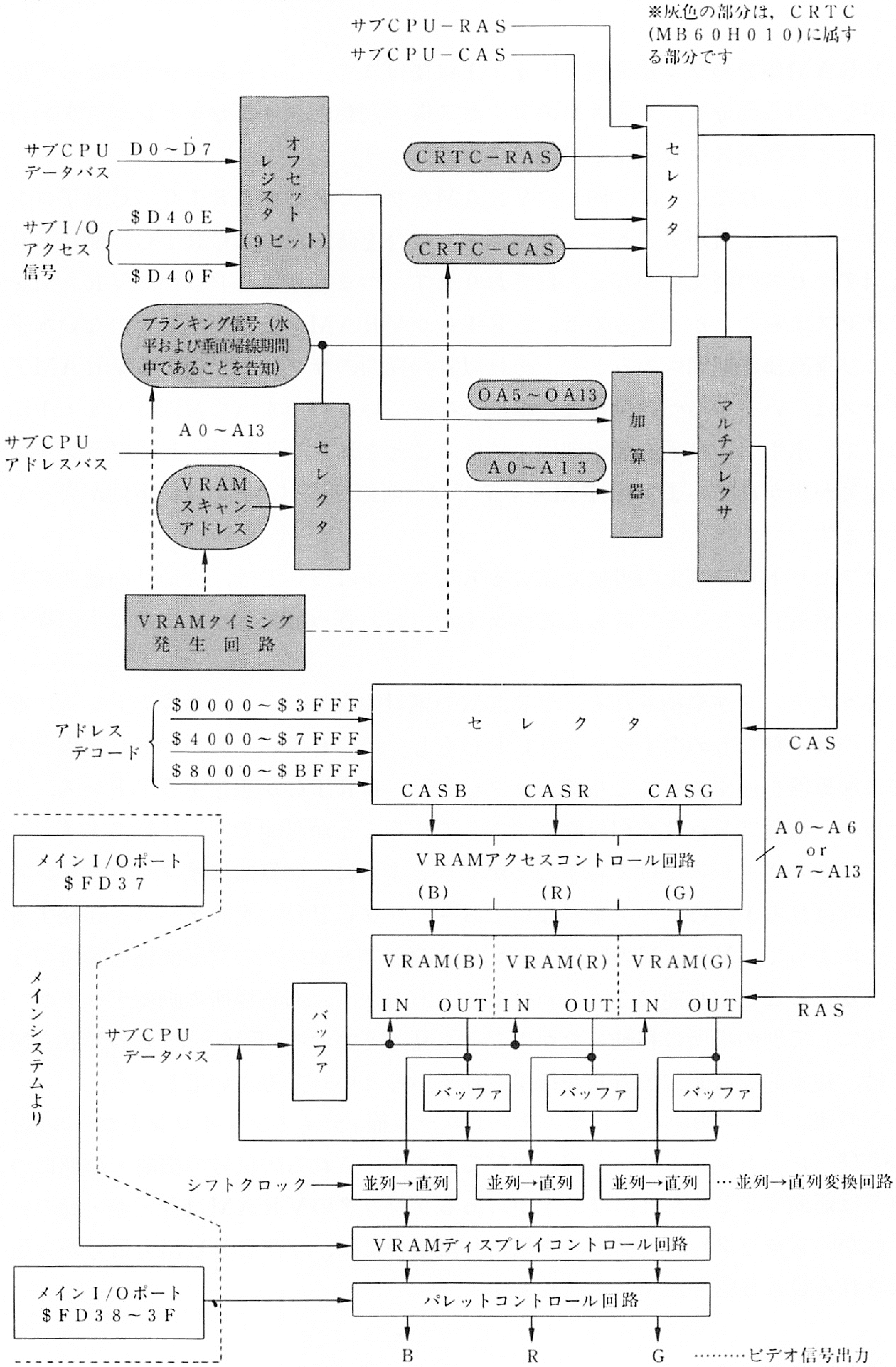
前節でもふれたように，同一のＶＲＡＭをサブＣＰＵとＣＲＴＣ（ＣＲＴコントローラ）の２者がアクセスする関係上，競合を防ぐためにＣＲＴＣの側にＶＲＡＭアクセスの優先権が与えられております．つまりサブＣＰＵが，ＶＲＡＭをアクセスすることができるのは，ＣＲＴＣがＶＲＡＭを読み出す必要のない水平および垂直帰線期間中のみとし，それ以外の期間のサブＣＰＵによるＶＲＡＭアクセスは，いっさい受け付けない機構になっているのです（なお図２・４・１において，水平および垂直帰線期間中であることを知らせる信号としてブランキング信号が描かれています）．ＦＭ－７７では，前節で述べたとおり，事情が異っております．

オフセットレジスタの書換えによるスクロールについては，次節「高速スクロールの構造」のところで詳しく述べますが，おおざっぱに言うと次のようになります．

各々のデータが格納される，ＶＲＡＭの絶対的なアドレス（物理アドレス）そのものは不動なものですが，サブＣＰＵもしくはＣＲＴＣからのアドレス線との間に加算器を通すことによって，サブＣＰＵ・ＣＲＴＣが想定するアドレス，すなわち，論理アドレスを相対的にずらしてやることが可能です．任意のタイミングで任意のアドレスだけずらすことができるように，加算器にオフセットレジスタと呼ばれるＩ／Ｏポートをつないでおき，サブＣＰＵのデータバスと連絡することによって，ＶＲＡＭの物理アドレスと論理アドレスとの対応関係を瞬時のうちに変えることが可能になったわけです．あたかも，ある場所の画面データが，一瞬にして別の場所に移動したかのごとく見えるのです．ＦＭ－7/8のＶＲＡＭには，初歩的な仮想記憶方式が採用されているといってもよいでしょう．

この他，メイン側からアクセスコントロール線，ディスプレイコントロール線，およびパレットコントロール線が伸びてきます．これらの信号の機能・形態については前節でまとめたとおりですが，あるブロックのＶＲＡＭ（青・赤・緑のいずれかのブロック）のアクセスを禁止した場合には，サブＣＰＵ出力信号から生成されるＣＡＳ信号がマスクされます．

図2・4・1　VRAMブロック図



2－5　高速スクロールの構造

2－5－1　表示されない部分とオフセットレジスタ

1）ＶＲＡＭに対する計算

ＦＭ－７／８では，１画面に対するＶＲＡＭは１６３８４バイト（＄４０００バイト）割り当てられています．ここで，１画面に何バイト必要であるかを正確に計算してみると次のようになります．

６４０×２００÷８＝１６０００（＄Ｅ８０）

８で割っているのは，１バイトにつき８ドットを表すことができるからです．これから考えれば，

１６３８４－１６０００＝３８４（＄１８０）

バイトの余分な領域が存在することになります．この３８４バイトというのは，３０７２ドット，すなわち４．８ライン（１ライン＝６４０ドット）に相当します．４．８ラインとは，６４０ドット×４ラインと５１２ドットに対応します（図２・５・１）．これだけの領域が，表示されない部分として残されていることを覚えておいて下さい．なお，ＶＲＡＭの具体的な構成については，３－５節を御覧下さい．

図２・５・１　表示されない領域

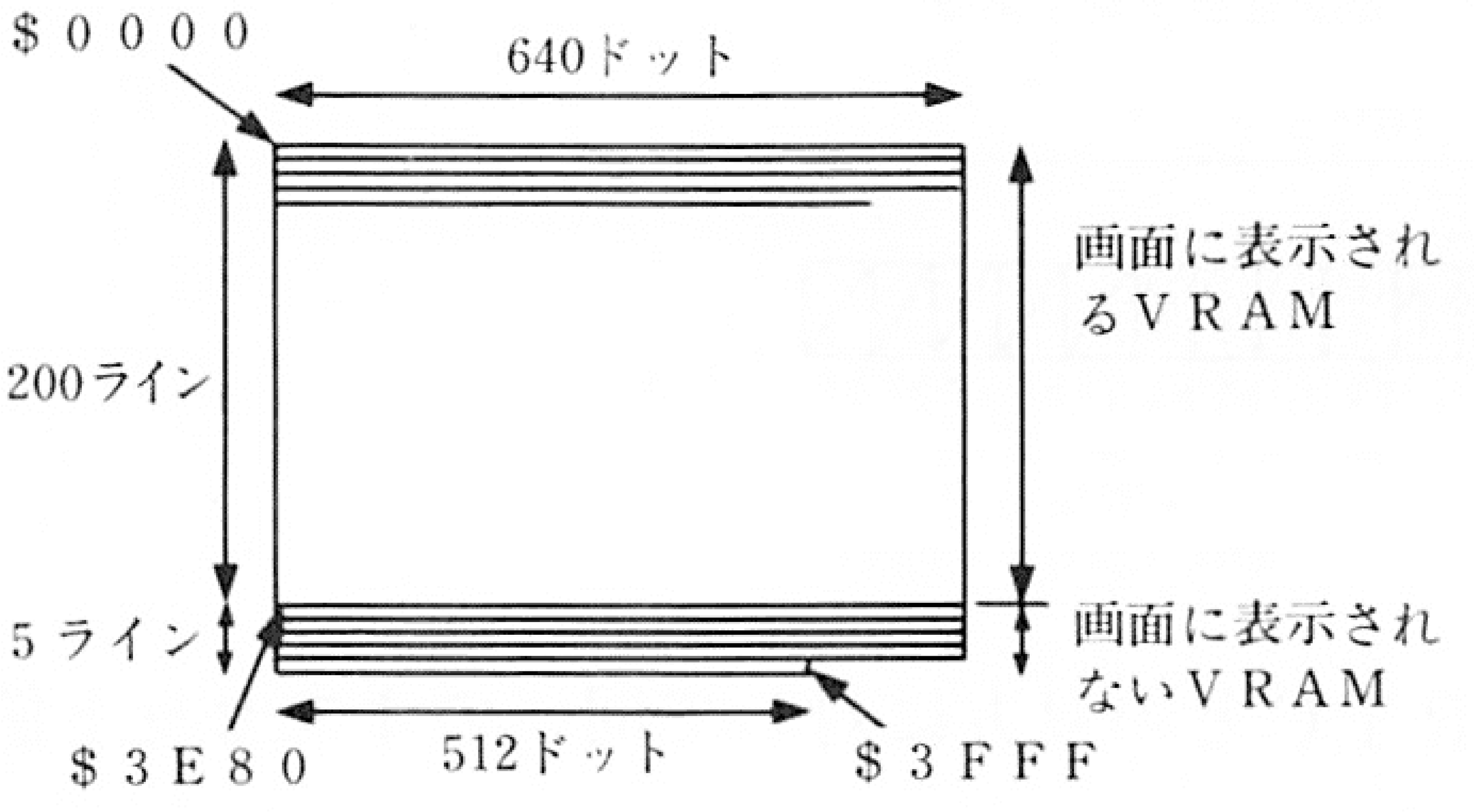
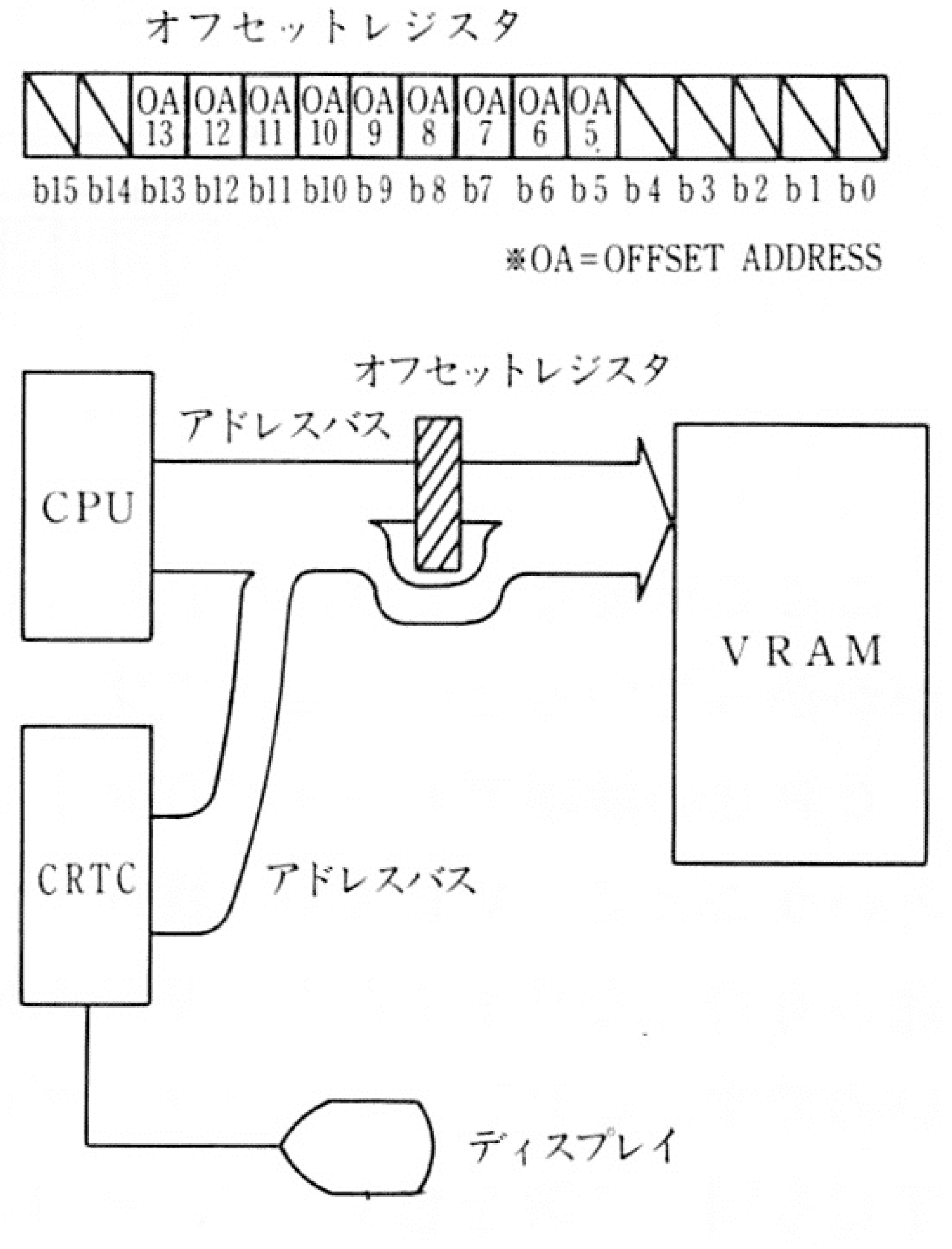


図２・５・２　アドレスとオフセットレジスタ



2）オフセットレジスタ

FM－7／8では，ハードウェア的にスクロールを速くするためのオフセットレジスタが設けられています．スクロールに関する話題というのは，このオフセットレジスタについての知識ということになりますので，これからはオフセットレジスタを中心に話を進めることにします．

オフセットレジスタは，CPUとCRTCがVRAMをアクセスする途中に設けられており，VRAMの実際のアドレス（物理アドレス）は，サブCPUやCRTCが出力したアドレス（論理アドレス）に，オフセットレジスタの値が加算された値になります．すなわち，

（VRAMの論理アドレス）

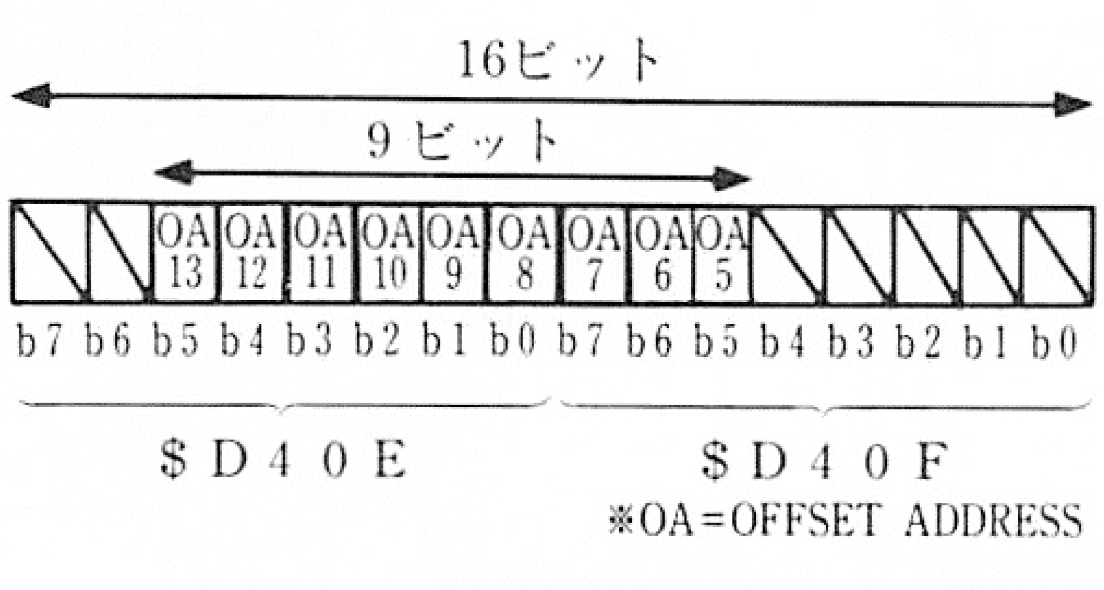
＝（CPU・CRTCが出力した論理アドレス）＋（オフセットレジスタの値）

そのため，オフセットレジスタの値を変化させることによって，CPU・CRTCがアクセスするVRAM物理アドレスは変化しますので，画面上のデータは瞬間的に移動（スクロール）したように見えるのです．

・例えば，オフセットレジスタが0のとき，VRAM上の物理アドレス＄0100番地に＄80というデータ（ドット）が書かれていたとします．

オフセットレジスタを0から＄00A0に換えた場合には，先程の＄0100番地は，CPU，CRTCから見れば＄0060番地(上記の式で言えば，＄0100＝＄0060＋＄00A0）となり，従ってCRTCによって＄0060番地に対応する位置へそのデータが表示されます．

図2・5・3　オフセットレジスタ



ここで重要なことは，CPUとCRTCとの双方に対して同一のオフセット値がかかっているということです．

CPUの論理アドレス＝CRTCの論理アドレス

ということで，VRAMの物理アドレスとの相対関係は，CPUとCRTCで共通のものとなりますから，VRAMの物理アドレスを全く考慮に入れなくてもよいのです．CPU（つまりユーザ）が想定した位置にドットを書き込むと，CRTCも同じ位置を想定して，そのドット表示をしてくれるのです．だから，普段の状態ではVRAMのどこが使用されているかを全く無視してよいのです．

２－５－２　スクロール

スクロールの場合も，ＶＲＡＭの物理アドレスは無視してよいので，これからはそのことについていっさいふれません．ＶＲＡＭは，ＣＰＵ・ＣＲＴＣが想定している論理アドレスに基づいて記述します．そして，ユーザの皆さんは次の２つを記憶しておいて下さい．

・スクロールは，オフセット値を変更したときの相対的な移動である．

・ＣＰＵ・ＣＲＴＣの論理アドレス＄３Ｅ８０～＄３ＦＦＦ（１画面）の領域にあるＶＲＡＭは表示されない．

なお，オフセットレジスタのオフセット値は，青，赤，緑の３画面に対して一様にかけられますので，簡単のため１画面（青の画面）に対してのみ説明を行っていきますが，同時に同じことが他の画面についても行われていることを忘れないで下さい．

１）オフセット値による相対移動

オフセットレジスタは，１６ビットありますが，その中で有効なのはビット５～１３の９ビットだけです．オフセットをかけるにあたって，１番小さい正のオフセット値から列記していくと，

＄００００，＄００２０，＄００４０，＄００６０，＄００８０，＄００Ａ０，
＄００Ｃ０，＄００Ｅ０，＄０１００，＄０１２０，＄０１４０，…………

ということになります．また，オフセット値による相対移動を考えるときは，元のオフセット値というものはあまり重要でなく，今回そのオフセット値にどれだけの値を足した（引いた）かということ（オフセットの増分）が重要になります．

１番小さい正のオフセットである＄００２０を足した場合についてみます．式で表すと（ＯＦＦＳＥＴはオフセットレジスタの値）次のようになります．

ＯＦＦＳＥＴ　＝　ＯＦＦＳＥＴ　＋　＄００２０

このとき，画面は図２・５・４のように左に２５６ドット移動することがわかります（ＶＲＡＭが連続しているため，正確に左に移動しているのではなく，元の画面の左２５６ドットは１ドット上に上がった形で画面の右側に表示されます）．

２５６ドット左にスクロールするのは，

＄００２０　×　８（ドット）　＝　２５６（ドット）

ということになります．例えば，今まで論理アドレス＄００２０番地にあったデータが，オフセットをかけた後は論理アドレス＄００００番地に移ります．

また，今まで＄０００１Ｆ番地にあったデータは，画面に表示されない領域である＄３ＦＥ０～＄３ＦＦＦ番地に移動しています（図の斜線部分の移動がそれにあたります）．これを式で表してみると，＄００００にあったデータは，

＄００００　－　＄００２０　＋　＄４０００　＝　＄３ＦＥ０

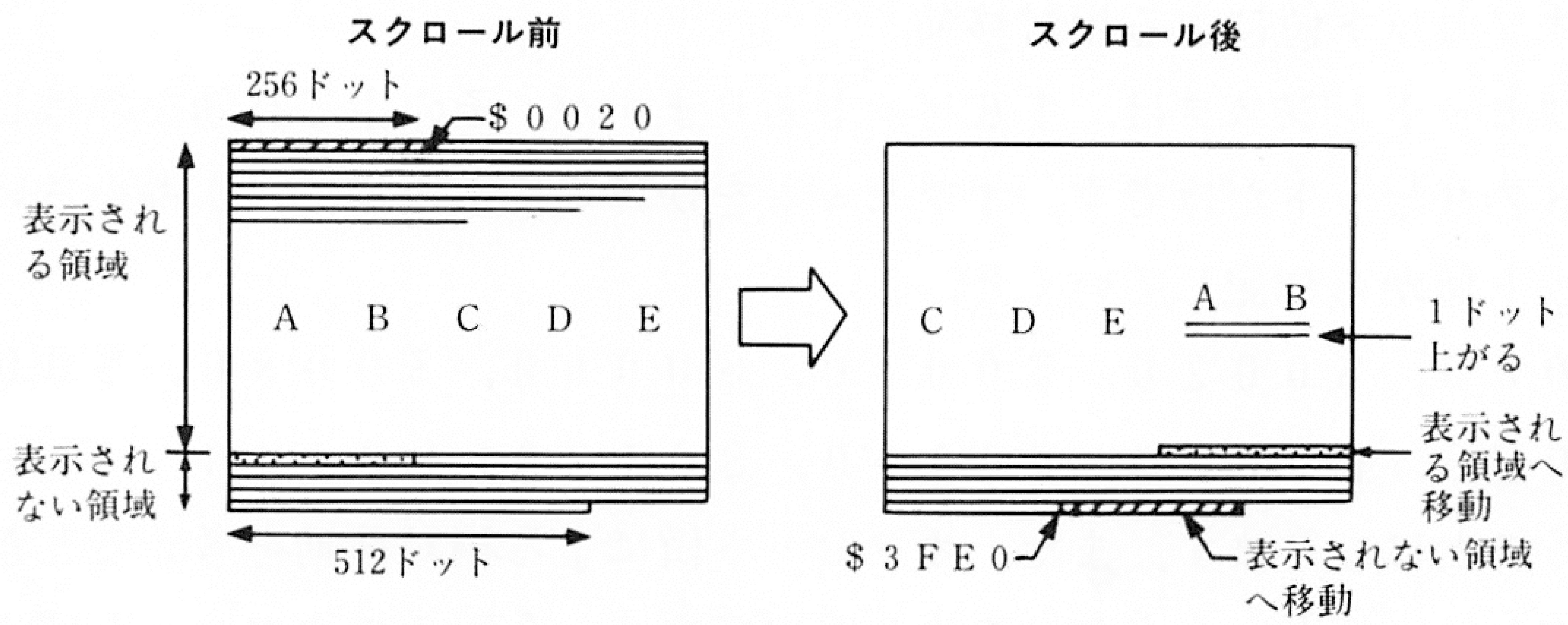
ということで＄３ＦＥ０に移ります．＄４０００を足しているのは，画面が循環しているため（オフセットレジスタでは，ビット１４以上は無視するため）です．

先程のように，スクロール後のアドレスを計算する式は次のようになります．

（スクロール後のアドレス）＝（スクロール前のアドレス）－（オフセット増分）

＋ { 上の部分の値が正のときは　＄００００
　　上の部分の値が負のときは　＄４０００ }

図２・５・４　＄００２０を足したときのスクロール



同様にオフセットとして＄００４０を足したときは，図２・５・５のように５１２ドット左に(１ドット上がって１２８ドット右に)移動したことになります．斜線の部分は＄００００～＄００３Ｆ番地で，表示されない領域へ移動する部分です．

＄００６０を足したときは，１ドット上がって左に１２８ドット移動したことに，＄００８０を足したときは，１ドット上がって右に２５６ドット移動したことになり，＄００Ａ０を足したときは，上に2ドット移動したことになります．

それ以上は，同じような繰返しになります．また，引いた（負の値を足した）場合は，今まで述べたことの逆のこと（元の画面がオフセットをかけた後の画面と入れ替わる）が起こります．例えば，＄００Ａ０を引くと下に2ドット移動します．

また，説明では述べませんでしたが表示されない領域に移動した分だけ，表示されない領域から表示される領域に移った部分があることも忘れないで下さい．

図2・5・5　＄００４０を足したときのスクロール

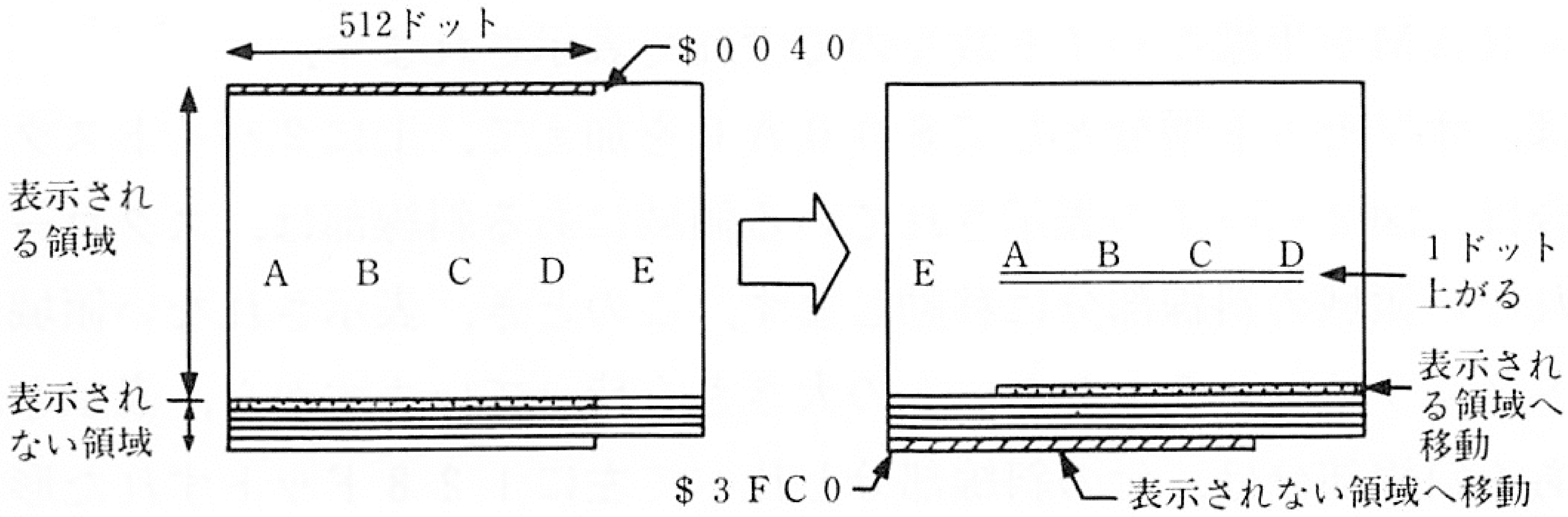
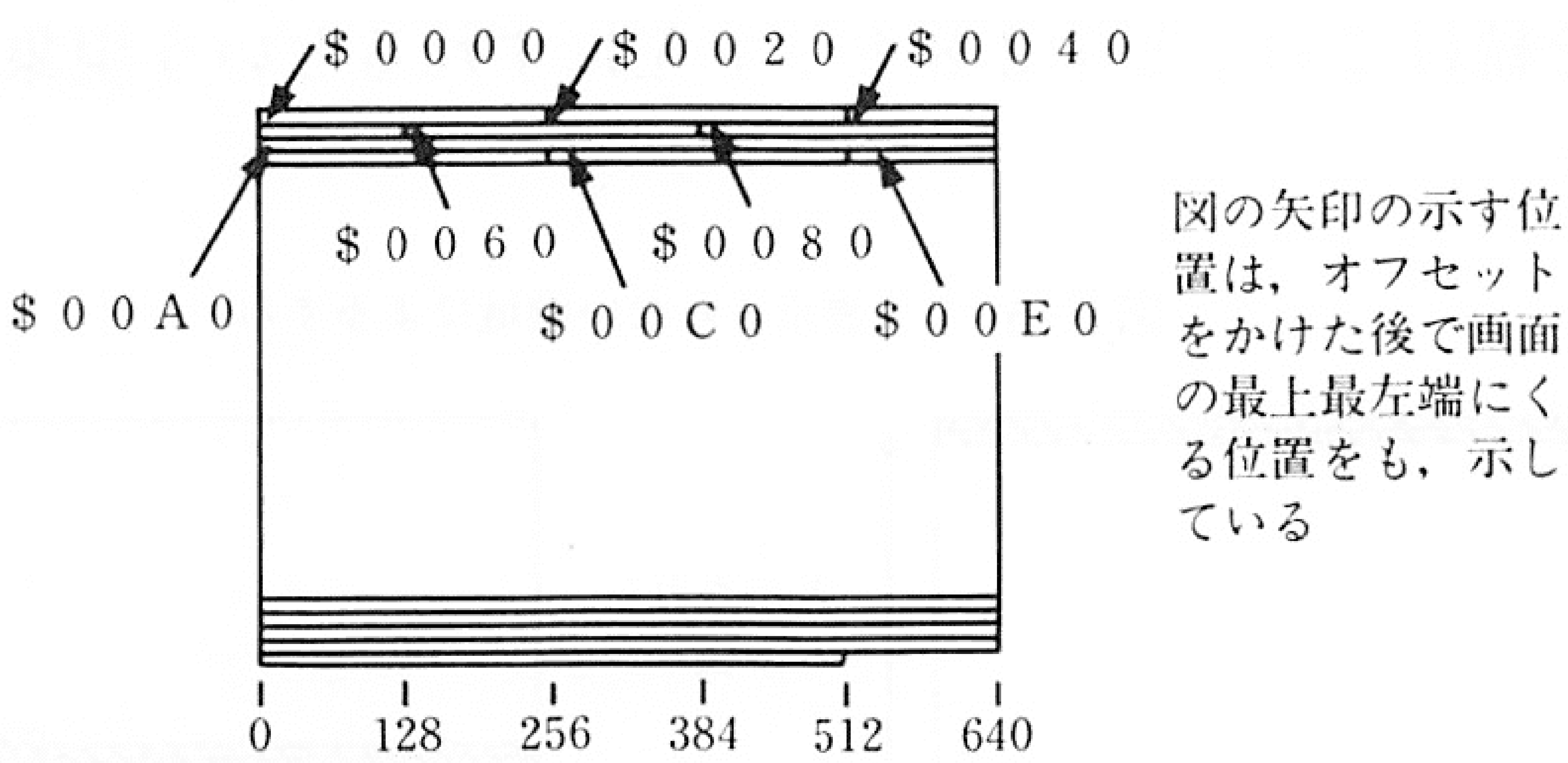


図2・5・6　スクロールとアドレス



オフセット増分に対しては，次のような公式ができます（負のときは逆になります）．￥は整数除算を，ｍｏｄは整数除算の剰余の演算を表します．

((オフセット増分)　￥　＄００Ａ０)×２　＝　(縦方向の移動ドット数)

((オフセット増分)　mod　＄００Ａ０)が，＄００００のときは，横方向の移動なし，＄００２０のときは左に２５６ドット移動，＄００４０のときは上に１ドット，右に１２８ドット移動(左へ５１２ドット移動)，＄００６０のときは上に１ドット，左に128ドット移動，＄００８０のときは上に１ドット，右に２５６ドット移動します．

このようにＦＭ－７/８では，縦方向には最低２ドット，横方向には最低１２８ドットずつしかスクロールできません．

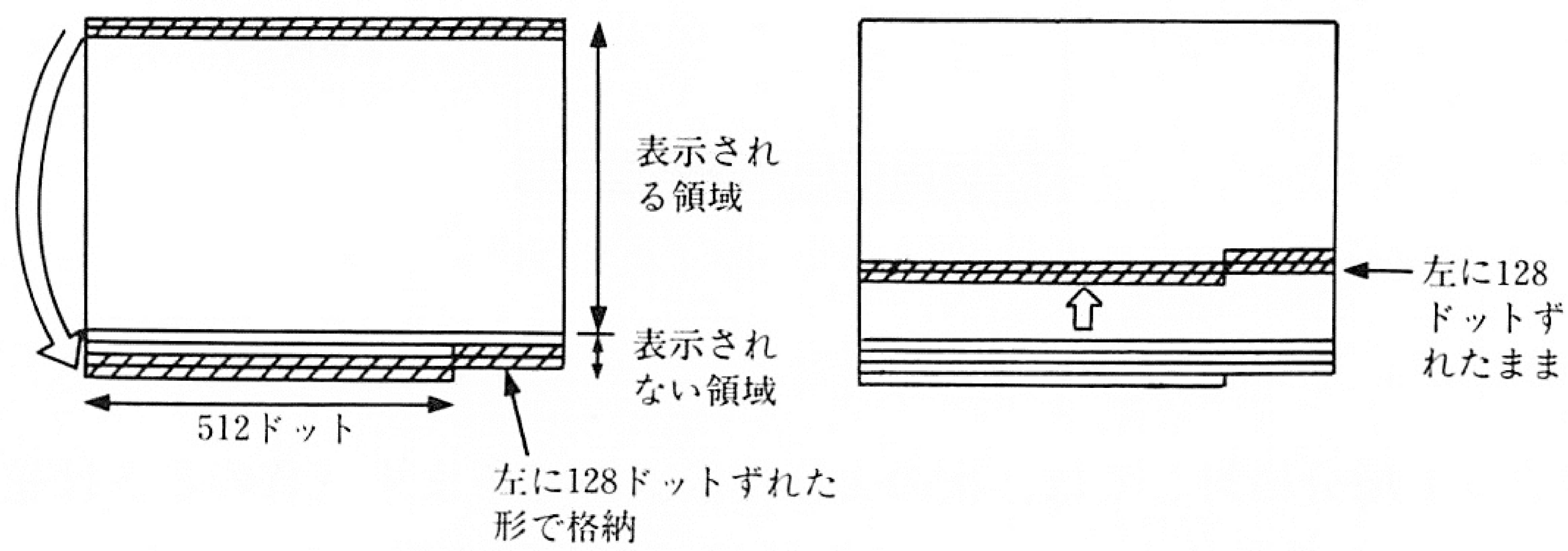
２）表示されない領域

スクロールを続けると，表示されない領域に入った部分が，再び表示される領域に移動してきます．このとき，元の形のまま表示されれば問題はないのですが，実際はＶＲＡＭが半端なバイト数なのでずれて表示されます．

例えば，オフセット増分として＄００Ａ０を加えて，上に2バイトスクロールした場合は，図2・5・7の表示されている領域にある斜線部は，スクロール後，表示されない領域の斜線部分に移動します．このとき，表示されない領域が６４０ドット×４ラインと５１２ドットの大きさを持っていますから，表示されない領域にある斜線部分は，元の斜線部分と比べて左に１２８ドットずれた形で格納されたことになります．だから＄００Ａ０を加え続けて，表示される領域に移動してきたときは，図2・5・8のように左に１２８ドットずれた形で出てきます．

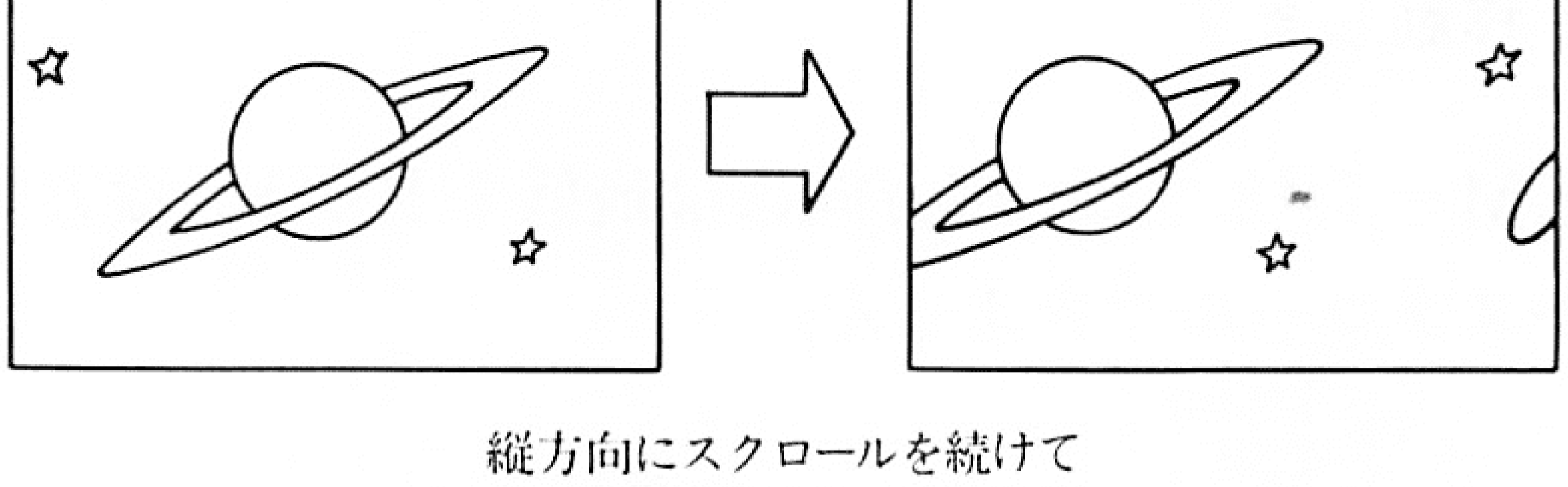
このように，５１２ドットという半端な大きさを持っているため，縦方向にスクロールを続けていくと１２８ドット左にずれるという現象が起こります（図2・5・7参照）．

図2・5・7　表示されない領域によるずれ



このことを考慮に入れれば，先程のオフセット増分の公式において，増分が＄００Ａ０で割り切れても，横にずれる（スクロールする）可能性があるので注意して下さい．

図2・5・8　ずれから起こる現象



縦方向にスクロールを続けて
元のＹ座標に戻ったときは
左に128ドットずれている

3）スクロール実行法

Ｉ／Ｏポートのオフセットレジスタは書込みだけが可能で，読み込もうとすると値が不定になります．そこで，サブシステムのモニタはオフセット値を保存するためのワークエリアとして，.ＶＯＦＳＴというラベルの２バイトのワークエリアを設けています．ＦＭ－７では，＄Ｄ０１Ｆ～＄Ｄ０２０番地がそれにあたり，ＦＭ－８では＄Ｄ０１Ｄ～＄Ｄ０１Ｅ番地がそれにあたります．

両方の機種に使えるように，ラベル名を使用してアセンブラで書くと，以下のようなプログラムになります．＄Ｄ４０Ｅ～＄Ｄ４０Ｆ番地がオフセットレジスタのあるＩ／Ｏポートで，それに書き込むときには，必ずＶＲＡＭアクセスフラグ（＝＄Ｄ４０９）をＯＮにしておいて下さい．

```
 LDD   .VOFST························元のオフセット値をとり出す
 ADDD  OFFSETVALUE··················オフセットの加算値を足す
(SUBD  OFFSETVALUE)                  (あるいは引く)
 BITA  $D409························ＶＲＡＭアクセスフラグＯＮ
 STD   .VOFST························オフセット値をワークエリアに
 STD   $D40E·························オフセット値をオフセットレジスタに
 STA   $D409························ＶＲＡＭアクセスフラグＯＦＦ
```

2－5－3　スクロール活用法

1）上，下スクロールでずれなくする方法

上(下)方向のスクロールで，表示されない領域の最後が５１２ドットであるために，スクロールを続けると１２８ドット左にずれてきますが，これを防ぐ方法があります．ただし，この方法は，２ドットあるいは4ドットの上下方向のスクロールにしか使用できません．

２ドット上にスクロールする場合は，オフセットをかける前に，図2・5・9のように画面最上の２ラインの内容を，表示されない領域の上２ラインに転送しておきます．この後，オフセットをかけて（＄００Ａ０を足して）スクロールを行うと，表示される領域の下２ラインに出てくるのは，スクロールをかける前の画面最上の２ラインの内容になります．

２ドット下にスクロールする場合も，図2・5・9に示すように表示される領域の下２ラインの内容を，表示されない領域に転送してからスクロールをすると，画面最上には元の画面の下２ラインの内容を表示させることができます．

なお，転送するときは，青，赤，緑の３画面について行って下さい．４ドットの場合も同様の転送を４ラインにして行って下さい．

図2・5・9　ずれをなくす方法

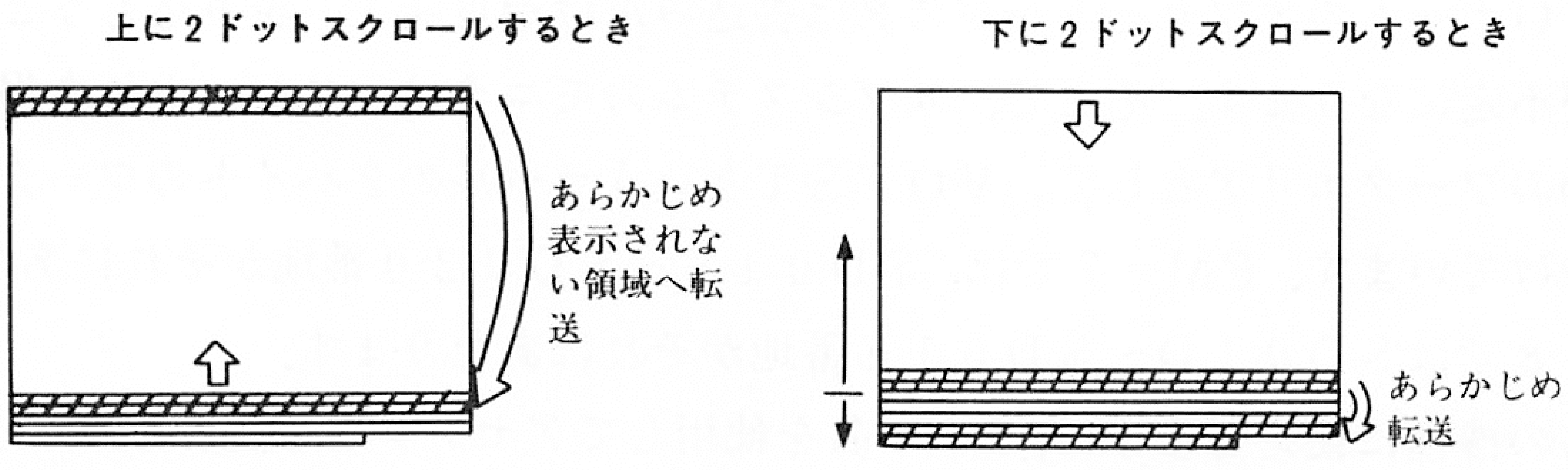
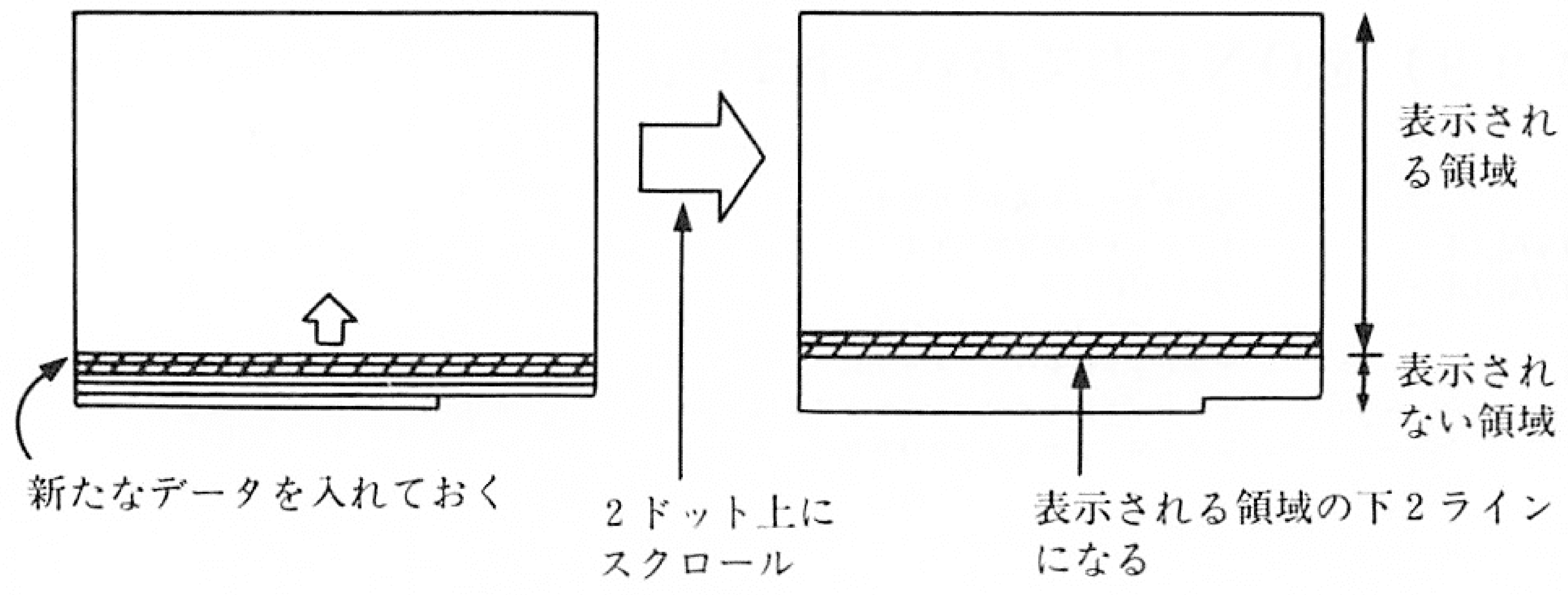


図2・5・10　新たなデータを出現させる方法



さらに，図2・5・10のように表示されない領域を，新たなデータを格納してからスクロールすれば，見ている者にとっては非常にきれいなスクロールとして映るので，スクロールを利用したゲームなどに応用できます．

2ドット上にスクロールするときのプログラムを，この節の最後に掲載しましたので御参照下さい．

2）オフセット値のワークエリアＶＯＦＳＴ

Ｉ／Ｏポートのオフセットレジスタは9ビットだけが有効でしたが，ワークエリアの．ＶＯＦＳＴは１６ビットが有効になります（ただし，この中の9ビットだけがＩ／Ｏポートに書き込まれるときに意味を持つのですが）．これを利用すると，オフセットに半端な数値を足していってやれば，何回かに一回は違うスクロールを行うことができます．

例えば，＄００Ａ１をオフセット増分として足していった場合，ワークエリア．ＶＯＦＳＴには＄００Ａ１の有効でない＄１という値も足されていきます．このスクロールを３２回行うと，３２回目には，（＄１×３２）＝＄２０となり，２ドット上にスクロールをすると同時に，左に２５６ドットスクロールします．同様に，＄００Ａ２のときは１６回に１回，＄００Ｂ０のときは2回に１回の割合で左に２５６ドットスクロールを行います．また，引いていく場合も同様のこと

が言えます（例えば，＄００Ａ２を引いていけば，１６回に１回は右に２５６ドットスクロールします）．

```
                               NAM    2DOTUP
                               OPT    MEM,NOGEN
       5000                    ORG    $5000
                        *
               D01F     .VOFST EQU    $D01F    * $D01D on FM-8
                        *
                        *  move to shared ram
                        *
               5000     START  EQU    *
       5000 B6 FD05            LDA    $FD05    * HALT to subsystem
       5003 2B FB   5000       BMI    START
       5005 86 80              LDA    #$80
       5007 B7 FD05            STA    $FD05
                        *
       500A 30 8D 0010         LEAX   PROGRM,PCR
       500E CE FC80            LDU    #$FC80
       5011 C6 42              LDB    #ENPROG-PROGRM
               5013     RP1    EQU    *
       5013 A6 80              LDA    ,X+      * move program
       5015 A7 C0              STA    ,U+
       5017 5A                 DECB
       5018 26 F9   5013       BNE    RP1
                        *
       501A 7F FD05            CLR    $FD05    * go the program
       501D 39                 RTS
                        *
                        *  2 dot up scroll program
                        *
               501E     PROGRM EQU    *
       501E    0000            FDB    $0000    * TEST command
       5020    3F              FCB    $3F
       5021    59              FCC    'YAMAUCHI'
                        *
       5029    93              FCB    $93      * TEST subcommand
       502A    D38F            FDB    $D38F
       502C    90              FCB    $90
                        *
       502D B5 D409            BITA   $D409    * VRAM access on
       5030 8E 0000            LDX    #$0000   * poiter initialize
       5033 108E 3E80          LDY    #$3E80
       5037 C6 A0              LDB    #160
                        *
               5039     PROG1  EQU    *
       5039 A6 89 8000         LDA    $8000,X  * move GREEN VRAM
       503D A7 A9 8000         STA    $8000,Y
       5041 A6 89 4000         LDA    $4000,X  * move RED VRAM
       5045 A7 A9 4000         STA    $4000,Y
       5049 A6 80              LDA    ,X+      * move BLUE VRAM
       504B A7 A0              STA    ,Y+
       504D 5A                 DECB
       504E 26 E9   5039       BNE    PROG1
                        *
       5050 FC D01F            LDD    .VOFST   * set offset
       5053 C3 00A0            ADDD   #$00A0
       5056 FD D01F            STD    .VOFST
       5059 FD D40E            STD    $D40E
       505C B7 D409            STA    $D409    * VRAM access off
       505F 39                 RTS
                        *
               5060     ENPROG EQU    *
               5000            END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=505F
PROGRAM ENTRY ADDR=5000
```

2－6　タイマ用ＮＭＩ(ノンマスカブルインタラプト)

サブシステムでは，タイマ関係の処理を効率よく行うために，20ms毎に，ＮＭＩ割込みが加わるようになっています．このＮＭＩ割込みは，いかなるソフトウェア的手段によっても禁止することができません．ですから，ユーザがサブシステムを使いこなすにあたっては，常にＮＭＩがかかることを頭に置いておかなければなりません．例えば，ＮＭＩサービスルーチン内ではＤＰレジスタが参照されており，このＮＭＩはいつかかってくるかも知れないので，サブシステム内でユーザがＤＰレジスタの値を勝手に変えてしまうことは許されません．また，メイン側から依頼された割込み事象の発生は，ＮＭＩサービスルーチン内で，アテンション割込み端子を通じて明らかにされる機構になっているので，アテンション割込みを独自の目的で使用する場合など，特にこのＮＭＩのことを念頭に入れておく必要があります．

ＮＭＩサービスルーチンの内容は，タイマ関係の処理系サブルーチンと密接な関係にあるため，ここではふれずに第３章(３－７－５節)にまわします．ハードウェア的には，たいした問題はないと思います．

第3章

サブシステムのモニタ

モニタ内部ルーチンの全貌――

実際にサブシステムでプログラムを動かすにあたって，サブシステム上のモニタがどのように機能しているか，ワークエリアがどのように使用されているかがわかりませんと，非常に苦労することになります．逆にそれらの知識があれば，効率の良いプログラムができることになります．この章では，サブシステムのモニタの全サブルーチンについて系統的に解説することにしました．この章を参考に，ユーザの皆さんのプログラムの効率の向上について研究してください．なおアドレスはFM－7シリーズのものを用いています．

PHASE III
SUBSYSTEM

各ルーチンの書式の説明

コマンドルーチンについて

タイトル：コマンド名と，そのコマンドのコード，そのコマンドルーチンのエントリ（括弧でくくっています）を書きました．

機能のまとめ：コマンドの機能の簡単な説明です．

共有ＲＡＭ上のパラメータ：そのコマンドを呼び出すための，共有ＲＡＭ上のパラメータを表にまとめました．ただし，継続フラグビットとコマンドコードは省略しています．

共有ＲＡＭ上の復帰情報：そのコマンドを呼び出した後に，共有ＲＡＭ上に返される情報を表にまとめました．ただし，エラーコードと継続フラグビットは省略しています．

主なサブルーチン：そのコマンドの処理系において，重要であると思われるサブルーチンを列記し，簡単に説明しました．

補助ワークエリア：＄Ｄ０Ａ０以降のワークエリアは，各々のコマンドルーチンで独自の使われ方をしていますので，そのコマンドの処理系での使用法を説明しています．

解　　説：コマンドルーチンの解説です．

一般のサブルーチンについて

タイトル：最初にそのサブルーチンの代表的なエントリのラベルを掲げ，以下に各エントリについて，エントリアドレスおよび終了アドレスを書きました．ただし，終了アドレスが一致している場合は省略しました．

機　　能：サブルーチンの機能の簡単な説明です．

レジスタ：使用されているレジスタです．スタックに保存されるレジスタについても明記しました．ただし，ＰＣおよびＣＣについては省略しました．

入力情報：呼び出す前に与えなければならないパラメータです．

出力情報：呼び出した後，設定されるパラメータです．

ワークエリア：使用（書換え，参照）しているワークエリア，データです．

サブルーチン：使用しているサブルーチンです．ルーチンの終りでジャンプするという形で呼び出すものも含んでいます．

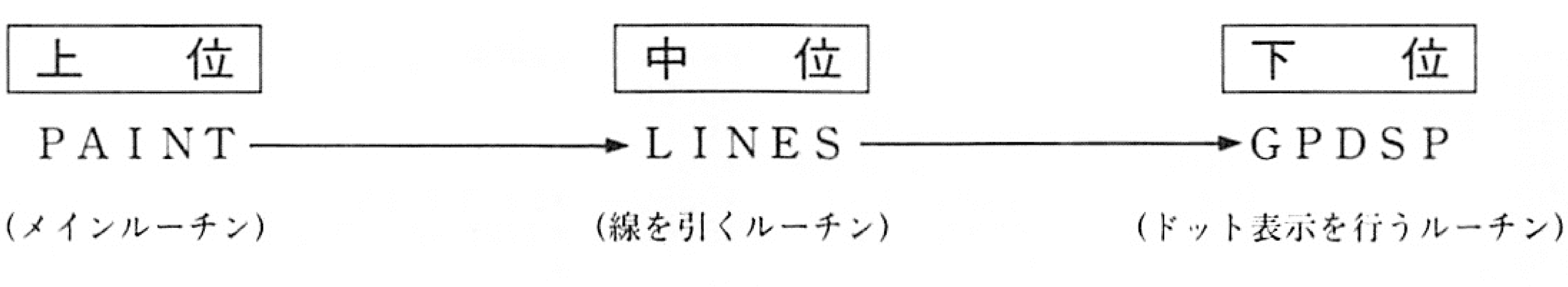
解　　説：簡単な説明です．

３－１　サブシステムのソフトウェア体系

ＦＭ－７/８のリセットベクトルは＄Ｅ０００番地に設定されており，サブＣＰＵの動作は＄Ｅ０００番地からのリスタートルーチンによって始まります．このリスタートルーチンによって，ワークエリアの初期化等を行った後，サブシステムはコマンド受付け状態となります．

サブシステムの動作は，コマンドハンドラもしくはコマンドコントローラと呼ばれるコマンド解釈ルーチン（メインループ）によって，すべて統轄されており，ＣＯＮＴコマンドを除いたすべてのコマンドの処理動作は，ここから始まります．コマンド解釈には，ジャンプベクトル方式が用いられており，各々のコマンド処理系エントリアドレスが書かれているジャンプテーブルから，コマンド毎のジャンプベクトルを読み取って，そのアドレスをコールします．このジャンプテーブルは，コマンドハンドラの直後（ＦＭ－７では＄Ｅ０７６～＄Ｅ０Ｃ６）に位置しております．コマンドハンドラに限らず，サブシステムでは至る所でジャンプベクトル方式が用いられており，サブルーチンの構造化とあいまって，整然としたアルゴリズムによるシステムソフトウェアの構築に一役買っております．

サブシステムの各処理系は，各コマンドのメインルーチン→中位のルーチン→最下位レベルの汎用サブルーチン群というように，構造化が進められていて，上位ルーチンから下位ルーチンまでほとんど数十バイトから百数十バイトの単位でまとめられており，効率よく運営されています．従って，ユーザがサブシステムを利用する場合には，必要なレベルでこれらの単位を呼び出すようにすると，スマートなプログラムを作成することができるのです．例えば，ＰＡＩＮＴ処理系に手を加えて改良したいような場合，なにも，ドット表示のサブルーチンからわざわざ作っていかなくても



という具合に続いているサブルーチン呼出しのチェーンのうち，自分の目的に合う適当なレベルの所で断ち切って呼び出せばよいわけです．

さて，サブシステムの各処理系は，類似の形体・機能を持っているルーチンごとにまとめられてＲＯＭ上に配置されているので，これらをブロックごとにまとめて，およそのメモリ割付けを図３・１・１に示します．

図3・1・1　FM-7のブロックごとのメモリの割付けの様子

アドレス	ブロックの種類
$E000~$E033	リスタートルーチン
$E034~$E06D	コマンドハンドラ（メインループ）
$E06E~$E075	IRQサービスルーチン
$E0CF~$E0F4	エラー処理系エントリ
$E0F5~$E14D	コマンド待ち・CONT処理のためのサブルーチン群
$E14E~$E382	INIT・ERASE系，バッファポインタの取扱い
$E2D3~$E5DF	下位レベルの汎用サブルーチン群（一部上記のものと重複）
$E5E0~$EE71	INIT・ERASEを除く，コンソール関係の処理系ルーチン群 (GET，PUT，Get Buffer Addressなど)
$EE72~$FD5B	グラフィック関係の処理系ルーチン群 （LINE，CHAIN，POINT，PAINTなど）
$FD5C~$FE7B	キー入力関係の処理系（FIRQサービスルーチンを含む）
$FE7C~$FF6E	タイマ関係の処理系（NMIサービスルーチンを含む）
$FF6F~$FFAD	TESTコマンド

さて，各コマンド処理系での処理動作が一通り終ると，最終的にRTS命令(もしくは，RTSと同等な命令）によって，解釈ルーチンへと戻ってきます．そして，BRA命令で解釈ルーチンの先頭へと再び入り，引き続き次のコマンドを受け付ける準備に入ります．これがメインループを形成しているわけです．

なお，各コマンド処理系内で，エラーが発生した場合には，該当するエラールーチンのエントリに分岐するようになっております．エラールーチンといっても，各エラーに対応しているエラーコードをエラーコードレジスタ（$D380）に代入して，解釈ルーチンの先頭へ戻るだけの簡単なものです．各エントリには「LDA　#エラーコード」の命令が書かれていて，これらのLDA命令が，「CMPX#（コード$8C)」によって次々に連結されている，という巧妙な構造から成り立っています（下記リスト参照)．

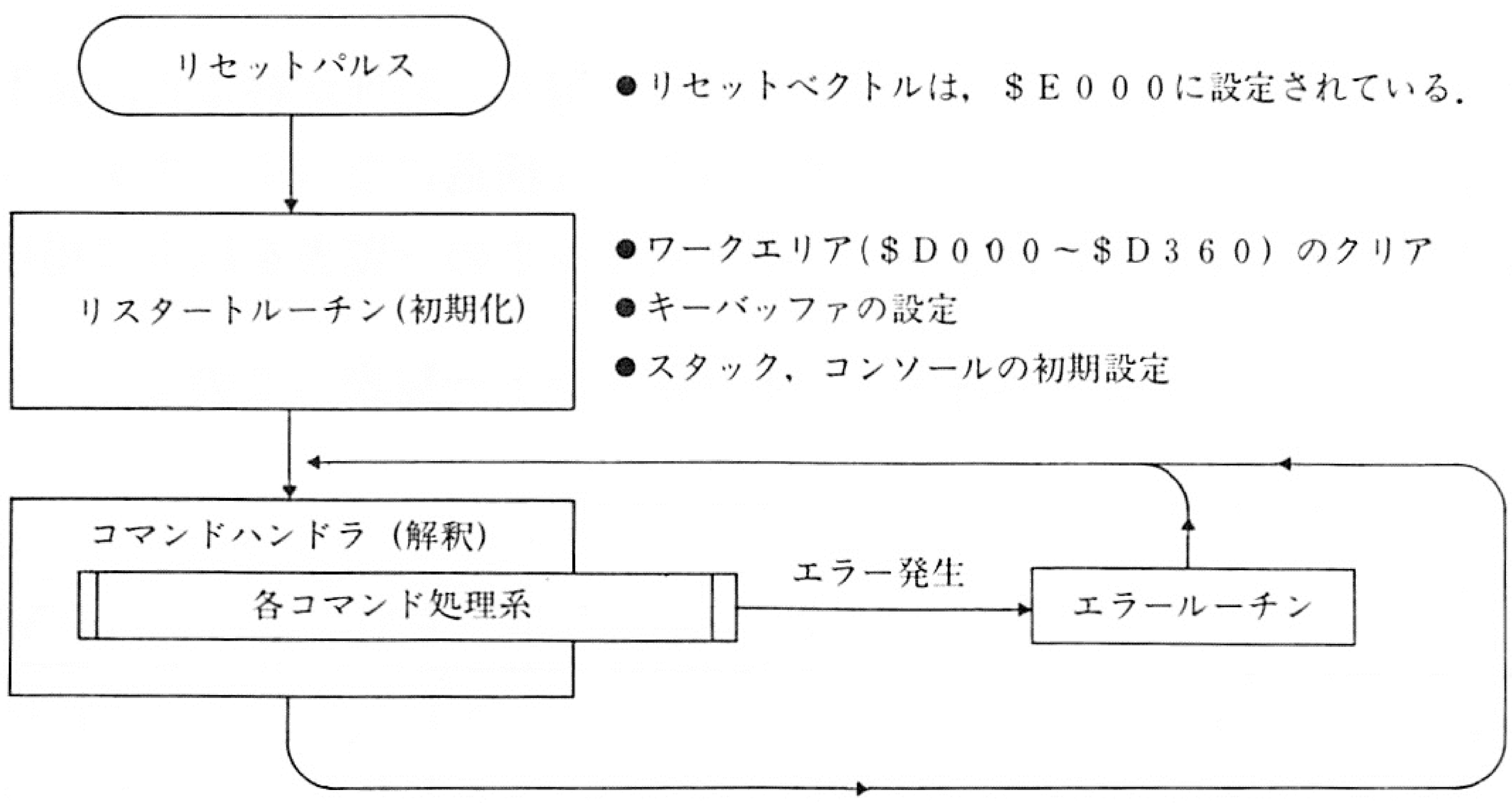
```
                                NAM    ERROR    ERROR  ﾉ ｼｮﾘ
                                OPT    NOMEM
      E0CF                      ORG    $E0CF
                00D0            SETDP  $D0
                D380    .COMAR  EQU    $D380    *ｴﾗｰｺｰﾄﾞ ﾚｼﾞｽﾀ
                E06C    CMSEND  EQU    $E06C    *ｺﾏﾝﾄﾞ ﾊﾝﾄﾞﾗ ｴﾝﾃﾞｨﾝｸﾞ
                        *                       *
                        *       ERROR ENTRY     *
                        *                       *
                E0CF    ERRENT  EQU    *
      E0CF 86   3C      ERR60   LDA    #60      *INITﾊﾟﾗﾒｰﾀ ｴﾗｰ
                        *Skip Next 2 Byte
      E0D1      8C              FCB    $8C      *CMPX#
      E0D2 86   3D      ERR61   LDA    #61      *ｺﾝｿｰﾙ ｻﾞﾋｮｳ ｴﾗｰ
                        *Skip Next 2 Byte
      E0D4      8C              FCB    $8C      *CMPX#
      E0D5 86   3E      ERR62   LDA    #62      *ｵｰﾀﾞｰ ｴﾗｰ
                        *Skip Next 2 Byte
      E0D7      8C              FCB    $8C      *CMPX#
      E0D8 86   3F      ERR63   LDA    #63      *ｸﾞﾗﾌｨｯｸ ｻﾞﾋｮｳ ｴﾗｰ
                        *Skip Next 2 Byte
```

```
E0DA       8C                  FCB     $8C        *CMPX#
E0DB 86    40          ERR64   LDA     #64        *ﾌｧﾝｸｼｮﾝ ｺｰﾄﾞ ｴﾗｰ
                       *Skip Next 2 Byte
E0DD       8C                  FCB     $8C        *CMPX#
E0DE 86    41          ERR65   LDA     #65        *ｻﾞﾋｮｳｽｳ ｴﾗｰ
                       *Skip Next 2 Byte
E0E0       8C                  FCB     $8C        *CMPX#
E0E1 86    42          ERR66   LDA     #66        *ﾓｼﾞｽｳ ｴﾗｰ
                       *Skip Next 2 Byte
E0E3       8C                  FCB     $8C        *CMPX#
E0E4 86    43          ERR67   LDA     #67        *ｶﾗｰ ｼﾃｲｽｳ ｴﾗｰ
                       *Skip Next 2 Byte
E0E6       8C                  FCB     $8C        *CMPX#
E0E7 86    44          ERR68   LDA     #68        *PFｷｰ ﾅﾝﾊﾞ ｴﾗｰ
                       *Skip Next 2 Byte
E0E9       8C                  FCB     $8C        *CMPX#
E0EA 86    45          ERR69   LDA     #69        *ｲﾝﾊﾞﾘｯﾄﾞﾊﾟﾗﾒｰﾀ ｴﾗｰ
                       *Skip Next 2 Byte
E0EC       8C                  FCB     $8C        *CMPX#
E0ED 86    46          ERR70   LDA     #70        *ｲﾝﾊﾞﾘｯﾄﾞｺﾏﾝﾄﾞ ｴﾗｰ
                       *
E0EF B7    D380        ERROR   STA     .COMAR     *ｴﾗｰｺｰﾄﾞ ﾉ ｶｸﾉｳ
E0F2 7E    E06C                JMP     CMSEND     *ｺﾏﾝﾄﾞﾊﾝﾄﾞﾗ ﾍ ｼﾞｬﾝﾌﾟ
                       *
                               END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=E0CF
PROGRAM END   ADDR=E0F4
PROGRAM ENTRY ADDR=****
```

以上の体系を簡単な概念図として，図３・１・２にまとめます．

図３・１・２　サブシステムのソフトウェア体系(概念図)



３－２　コマンド解釈ルーチンＣＭＤＣＴＬ
——サブシステム　メインループ——

前節で述べたとおり，リスタートルーチンによって初期化を終えると，サブシステムは，コマンドハンドラと呼ばれる解釈ルーチンに入って，コマンド受付け状態となります．実際にメイン側からのコマンド待ちを行うのは，ＣＭＤＷＡＩという下位ルーチンで，このルーチンはＣＯＮＴコマンド受付けの場合にも呼び出されます．図３・２・１にリスタートルーチン・メインループのフローチャートをまとめておきます．

先ほどもふれたとおり，コマンドの解釈・実行にはジャンプベクトル方式が用いられておりますが，コマンド解釈用のジャンプテーブルは，図３・２・２に示すように，三段構えの構造から成り立っています．つまりＦＭ－７では処理の内容によって，コマンドを４つのブロックに分類し，メイン側から要請されたコマンドが，４つのうちのどのブロックに属するかをまず最初に判定し，続いて，各ブロックに分かれているジャンプテーブルから，コマンドコードに対応するジャンプベクトルを，Ｘレジスタに読み込むわけです．この後「ＪＳＲ　,Ｘ」命令で各コマンド処理系を呼び出します．

各コマンド処理系では，エラーが発生しない限り，処理が終ると，ＲＴＳ命令などによって，メインループに戻ってくるような構造になっています．

それでは，各コマンド処理系のエントリポイントの一覧表を以下に掲げます．

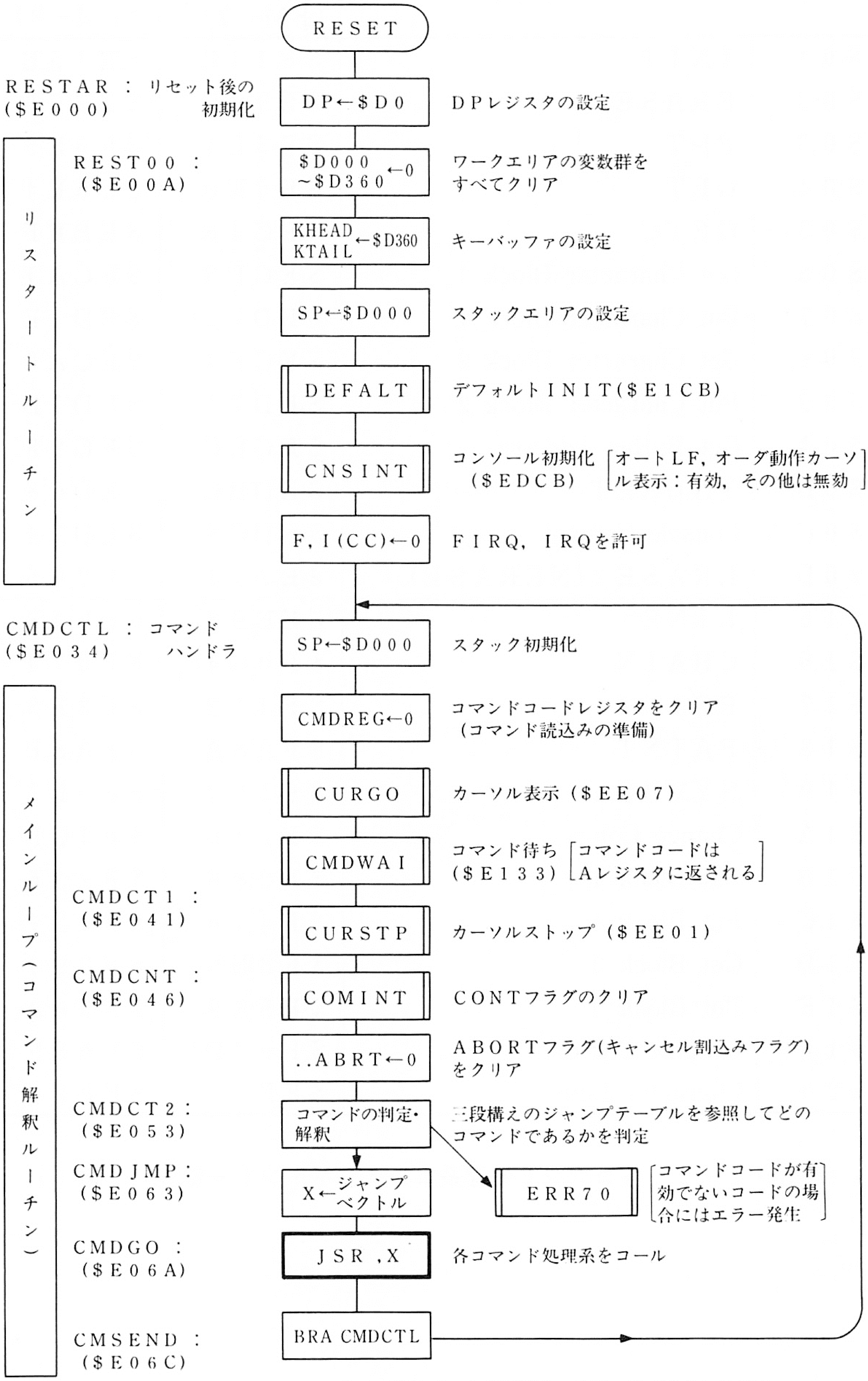
表３・２・１　各コマンド処理系エントリポイント一覧表（２の２）

コード	コマンド名	エントリポイント	
		（ＦＭ－７）	（ＦＭ－８）
＄２９	ＩＮＫＥＹ	＄ＦＤ５Ｃ	＄ＦＤ７６
＄２Ａ	Define String of ＰＦ.	＄ＦＥ０２	＄ＦＤＥ５
＄２Ｂ	Get String of ＰＦ.	＄ＦＥ４６	＄ＦＥ２９
＄２Ｃ	Interrupt Control	＄ＦＥ６１	＄ＦＥ４４
＄３Ｄ	Set Timer	＄ＦＥ７Ｃ	＄ＦＥ５Ｆ
＄３Ｅ	Read Timer	＄ＦＥＡ４	＄ＦＥ８７
＄３Ｆ	ＴＥＳＴ	＄ＦＦ６Ｆ	＄ＦＦ５２

コード	コマンド名	エントリポイント	
		(FM−7)	(FM−8)
$01	INIT	$E14E	$E15B
$02	ERASE	$E24B	$E250
$03	PUT	$E5E1	$E5E5
$04	GET	$E5E0	$E5E4
$05	GETC	$EC46	$EBEF
$06	Get Character Block 1	$ECF5	$ECA1
$07	Put Character Block 1	$ED72	$ED1E
$08	Get Character Block 2	$ECF4	$ECA0
$09	Put Character Block 2	$ED71	$ED1D
$0A	Get Buffer Address	$ECEC	$EC98
$0B	TABSET	$EDBC	$ED68
$0C	Console Control	$EDC8	$ED74
$0D	ERASE2(NERASE)	$E241	$E246
$15	LINE	$EE98	$EE48
$16	CHAIN	$F0C4	$F074
$17	POINT	$F412	$F3F3
$18	PAINT	$FA8A	$FA8B
$19	SYMBOL	$F103	$F0B3
$1A	Change Color	$F21B	$F1CB
$1B	Get Block 1	$F286	$F236
$1C	Put Block 1	$F446	$F427
$1D	Get Block 2	$F3B6	$F397
$1E	Put Block 2	$F589	$F56A
$1F	Graphic Cursor	$F67D	$F65E
$20	Character Line	$EE72	$EE22

表3・2・2　各コマンド処理系エントリポイント一覧表（2の1）

図3・2・1　リスタートルーチンとメインループ



※カッコ内に書いた16進数は，FM－7のアドレスです

図3・2・2　FM-7のコマンド解釈用ジャンプテーブルの様子

アドレス	内　容	解　説	
$E076	$04	ブロックの数	
$E077	$01	第1ブロック（コンソール系）に属するコマンドの	
	$0D	コマンドコードの範囲〔INIT～Console Control〕	
	$E087	第1ブロックのコマンドに対するジャンプテーブルの先頭番地	
$E07B	$15	第2ブロック（グラフィック系）	
	$20	〔LINE～Character Line〕	
	$E0A1		
$E07F	$29	第3ブロック（キー入力系）	
	$2C	〔INKEY～Interrupt Control〕	
	$E0B9		
$E083	$3D	第4ブロック（タイマ系，および，TESTコマンド）	
	$3F	〔Set Timer, Read Timer, TEST〕	
	$E0C1		
$E087	$E14E	INITコマンド（$01）の処理系エントリ	第1ブロック（コンソール系）
	$E24B	ERASE（$02）〃	
	⋮	⋮　⋮	
	$E241	ERASE2（$0D）〃	
$E0A1	$EE98	LINEコマンド（$15）の処理系エントリ	第2ブロック（グラフィック系）
	$F0C4	CHAIN（$16）〃	
	⋮	⋮　⋮	
	$EE72	Character Line（$20）〃	
$E0B9	$FD5C	INKEY（$29）	第3ブロック（キー入力系）
	$FE02	Define String of PF.（$2A）	
	$FE46	Get String of PF.（$2B）	
	$FE61	Interrupt Control（$2C）	
$E0C1	$FE7C	Set Timer（$3D）	第4ブロック（タイマ系 TEST）
	$FEA4	Read Timer（$3E）	
	$FF6F	TEST（$3F）	

ＣＯＮＴ

ＣＯＮＴＯ	＄Ｅ０Ｆ５	～	＄Ｅ１１４
ＣＯＮＴＩ	＄Ｅ１２１	～	＄Ｅ１３２
ＣＮＴＡＢＴ	＄Ｅ１１５	～	＄Ｅ１２０
ＣＯＮＴＯ２	＄Ｅ１０５		
ＣＯＭＩＮＴ	＄Ｅ１２Ａ		

機　能　メイン側に対する継続データ出力（ＣＯＮＴＯ）
メイン側からの継続データ入力（ＣＯＮＴＩ）　など

レジスタ　Ａ（保存）　※ＣＮＴＡＢＴの場合にはＳを再設定，他のレジスタ（Ｂ，Ｘ，Ｙなど）も使用

ワークエリア

ＥＲＲＲＥＧ	＄Ｄ３８０	エラーコードレジスタ	共有ＲＡＭ上のエリア
ＳＴＡＴＵＳ	＄Ｄ３８１	ステータスレジスタ	共有ＲＡＭ上のエリア
ＣＭＤＲＥＧ	＄Ｄ３８２	コマンドコードレジスタ	共有ＲＡＭ上のエリア
..ＣＯＮＴ	＄Ｄ００１	ＣＯＮＴフラグ保存用エリア	

※..ＣＯＮＴはＣＯＮＴＩ，ＣＯＭＩＮＴのみ使用

サブルーチン

ＣＭＤＷＡＩ	＄Ｅ１３３	コマンド待ちルーチン
ＣＵＲＳＥＴ	＄ＥＤＥ８	カーソル設定
ＣＭＤＣＮＴ	＄Ｅ０４６	メインループ中途エントリ

解　説　ＣＯＮＴＯではＣＯＮＴフラグ（ＳＴＡＴＵＳビット７）を立ててＣＯＮＴコマンド待ちを行い（ＣＭＤＷＡＩ），ＣＯＮＴコマンドのコード（＄６４）であれば，ＣＯＮＴＯ２に入る一方，ＣＯＮＴコマンドでない場合にはＣＮＴＡＢＴに入ります．

ＣＯＮＴＩではメイン側からの継続データを待つ（ＣＭＤＷＡＩ）のですが，ＣＯＮＴコマンドでなければ，やはりＣＮＴＡＢＴに入る一方，ＣＯＮＴコマンドであればＣＯＮＴＩ２に入って，ＳＴＡＴＵＳレジスタの内容を..ＣＯＮＴに格納した後，ＣＯＮＴＯ２で各レジスタのクリアを行います．

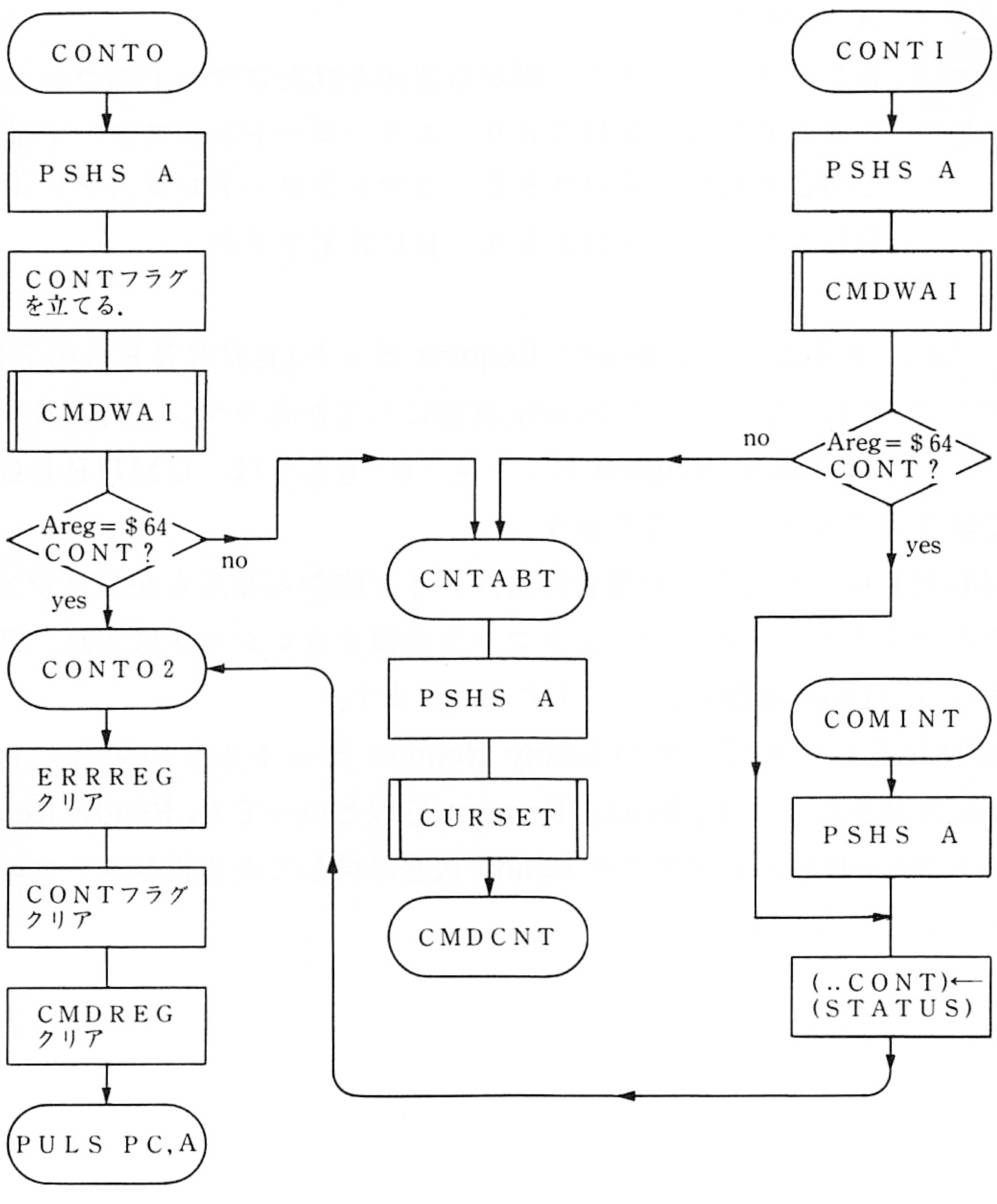
ＣＮＴＡＢＴはＣＯＮＴコマンド待ちの際に，メイン側からＣＯＮＴコマンド以外のコマンドが要請された場合，すなわちＣＯＮＴコマンドがキャンセルされたときの分岐先でスタックポインタＳ＝＄Ｄ０００に再設定後，カーソル設定ＣＵＲＳＥＴを行い，メインループの途中ＣＭＤＣＮＴに飛び込んで要請されたコマンドに対する処理を行います．

ＣＯＮＴＯ２は，ＣＯＮＴＯおよびＣＯＮＴＩのエンディングルーチンで，ＥＲＲＲＥＧ，ＣＯＮＴフラグ，ＣＭＤＲＥＧのクリアを行います．

COMINTは，CMDCTL（メインループ）から呼び出されるエントリで，Aレジスタのプッシュの後，CONTI2に入ります．

以上の様子をまとめたのが下図です．

CONT処理系の様子



ＣＭＤＷＡＩ

CMDWAI　　$E133 ～ $E14D

機　能	BUSYフラグをOFF（Ready状態）にしてメイン側からのコマンド待ちを行います．同時にReady Request処理をも行います．
レジスタ	A（保存）
出力情報	Aレジスタ　メイン側から要請されたコマンドのコード

ワークエリア			
ERRREG	$D380	エラーコードレジスタ	共有RAM上
CMDREG	$D382	コマンドコードレジスタ	共有RAM上
BUSY	$D40A	BUSYフラグ	

解　説　まずはじめにReady Requestビット（ERRREG第7ビット）をクリアして，BUSYフラグをReady状態にしてからコマンド待ちループに入ります．ループはReady Requestビット≠"0"もしくは，CMDREG≠0となるまで繰り返されることになります．

CMDREG≠0となった場合には，メイン側からなんらかのコマンドが送出されたわけですから，そのコマンドコードの値をAレジスタに入れ，BUSYフラグをON（Busy状態）にしてリターンします．

CMDREG＝0で，かつReady Requestビット≠0の場合には，Ready Request処理を行います．Ready Request処理といっても，Ready Requestビットをクリアし，BUSYフラグをReady状態に戻してから再びループの中へ入っていくだけのことです．

３－３　コンソール処理

３－３－１　ＩＮＩＴ　＄０１　（＄Ｅ１４Ｅ）

コンソール機能の初期化を行います．画面の桁数および行数や，各画面のサイズなどを設定します．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　　容	値
Ｄ３８３	ＢＣ	背景カラーコード	０～７
Ｄ３８４	ＮＣ	桁数	８０／４０
Ｄ３８５	ＮＬ	行数	２５／２０
Ｄ３８６	ＳＬ	スクロール開始行番号	０～(ＮＬ－１)
Ｄ３８７	ＮＳ	スクロール行数	０～(ＮＬ－ＳＬ)
Ｄ３８８	ＦＤ	ファンクションキー表示	ＦＤ＝０　表示しない ＦＤ≠０　表示する
Ｄ３８９	ＥＲＳ	初期設定後の画面消去	ＥＲＳ＝０　消去しない ＥＲＳ≠０　消去する
Ｄ３８Ａ	ＧＲ	単色表示(グリーン)	ＧＲ＝０　単色表示しない ＧＲ≠０　単色表示をする

主なサブルーチン

＄Ｅ１ＣＢ　ＤＥＦＡＬＴ　画面の初期化（デフォルト値による）
＄Ｅ１ＤＡ　ＨＯＭＥ　現在のウィンドウの決定
＄Ｅ２５Ｅ　ＥＲＳＡＬＬ　全画面の消去
＄ＥＤＥＦ　ＣＵＲＣＬＲ　カーソルのクリア

解　説　共有ＲＡＭ上のパラメータを読み取り，画面の情報のためのワークエリアを設定していきます．これらのワークエリアは，定数的，スイッチ的な性質を持っており，他のサブルーチンはこれらを参照して動作を行っています．よって，これらのワークエリアはこのコマンドルーチン以外では設定できません．

パラメータに誤りがあるときは，＄Ｄ３８０番地にエラーコード＄３Ｃを設定した後，ＤＥＦＡＬＴに入り，再びデフォルト値による画面の初期化を行います．

ワークエリア設定後は，ＨＯＭＥ（ＥＲＳ＝０のとき）あるいはＥＲＳＡＬＬ（ＥＲＳ≠０のとき）によって，それらのワークエリアの値に基づいた画面作り

を行います．

ＩＮＩＴ自体では，次のようなワークエリアを設定します．それぞれのワークエリアの詳細な説明は，必要なものについては随時行っていきます．

＄Ｄ０２２	ＰＵＴＣＦＧ	：ＰＵＴのときの継続フラグ　０に設定
＄Ｄ０２３	ＣＳＬＣＤＳ	：単色表示かカラー表示かを示すフラグ　ＧＲ＝０のとき０，ＧＲ≠０のとき＄ＦＦ
＄Ｄ０２Ｆ	..ＢＣＬ	：背景色カラーコード　ＢＣと同じ値
＄Ｄ０３１	..ＣＷＤ	：キャラクタ幅フラグ　ＮＣ＝８０のとき＄００，ＮＣ＝４０のとき＄０８
＄Ｄ０３２	..ＣＨＴ	：キャラクタアンダーマージンフラグ　ＮＬ＝２５のとき＄００，ＮＬ＝２０のとき＄０２
＄Ｄ０３６	..ＬＮＣ	：１行の桁数　ＮＣと同じ値
＄Ｄ０３７	..ＤＮＬ	：画面の行数　ＮＬと同じ値
＄Ｄ０３８	..ＦＳ	：ＰＦキー表示フラグ　ＦＤと同じ値
＄Ｄ０３９ ～Ｄ０３Ａ	..ＶＣＵ	：ＶＲＡＭ上の桁単位　ＮＣ＝８０のとき＄０００１，ＮＣ＝４０のとき＄０００２
＄Ｄ０３Ｂ ～Ｄ０３Ｃ	..ＶＬＵ	：ＶＲＡＭ上の行単位　ＮＬ＝２５のとき＄０２８０，ＮＬ＝＄０３２０
＄Ｄ０３Ｄ	..ＤＭＡＰ＋０	：スクロールモード画面開始行　ＳＬと同じ値
＄Ｄ０３Ｅ	＋１	：スクロールモード画面終了行　ＳＬ＋ＮＳ
＄Ｄ０３Ｆ	＋２	：ページモード１画面開始行　０に設定
＄Ｄ０４０	＋３	：ページモード１画面終了行　ＳＬと同じ値
＄Ｄ０４１	＋４	：ページモード２画面開始行　ＳＬ＋ＮＳ
＄Ｄ０４２	＋５	：ページモード２画面終了行　ＦＤ＝０のときＮＬと同じ値，ＦＤ≠０のときＮＬ－２
＄Ｄ０４３	＋６	：ＰＦキー表示画面開始行　ページモード２画面終了行と同じ値
＄Ｄ０４４	＋７	：ＰＦキー表示画面終了行　ＮＬと同じ値

ＤＥＦＡＬＴ

＄Ｅ１ＣＢ　～　＄Ｅ１Ｄ７

機　能	デフォルト値による画面の初期化を行います．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
ワークエリア	ＩＮＰＡＲＭ　＄Ｅ０Ｃ７～＄Ｅ０ＣＥ　初期化データ
サブルーチン	ＩＮＩＴ　＄Ｅ１４Ｅ　画面の初期化
	ＭＯＶＥ　＄Ｅ４４Ａ　ブロック転送

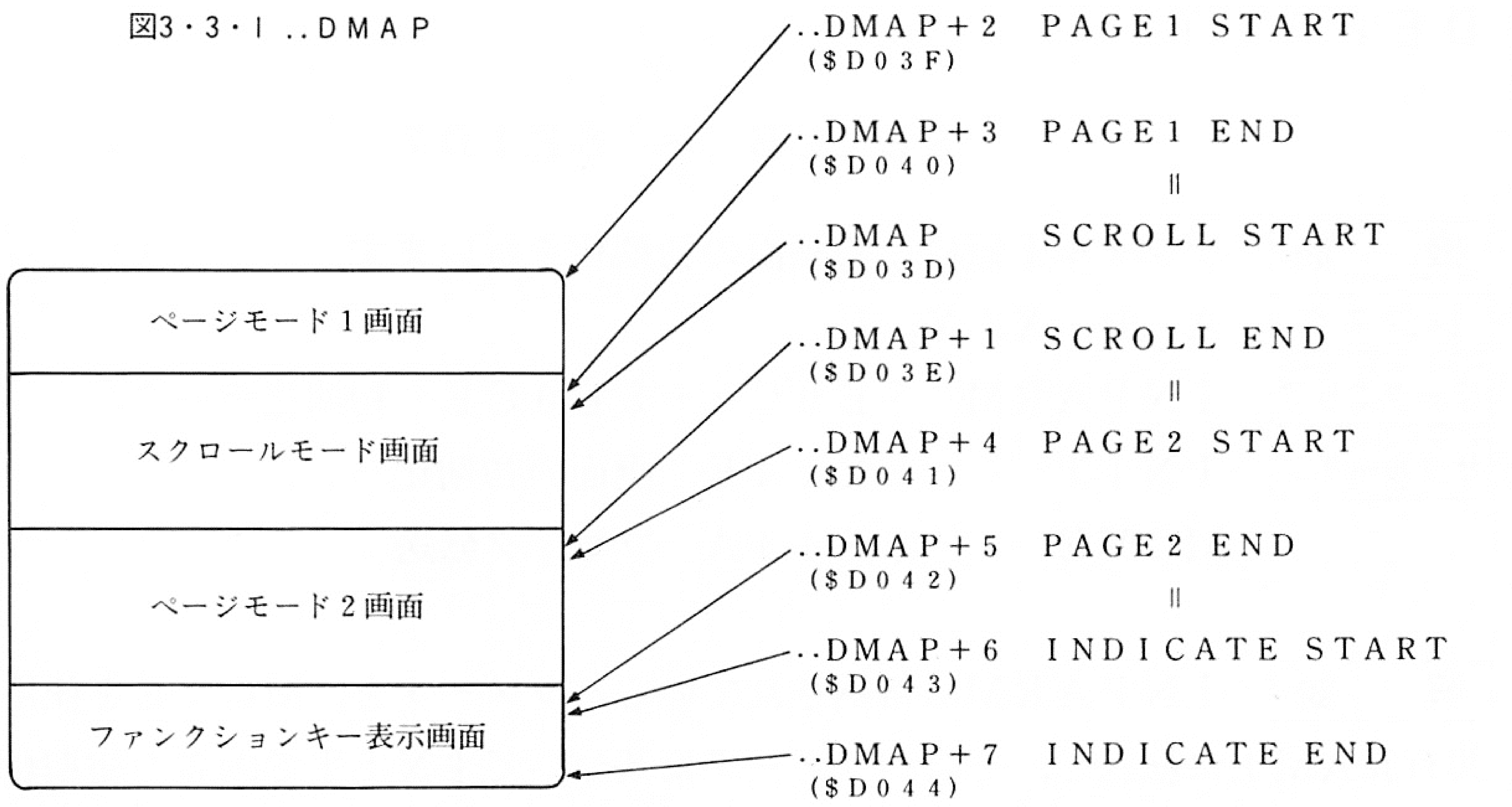
解　説　ＩＮＰＡＲＭに格納されているパラメータを，ＭＯＶＥを使用して共有ＲＡＭ上に転送し，そのパラメータに基づいてＩＮＩＴで画面の初期化を行います．共有ＲＡＭ上には，次のように転送されます．

＄Ｄ３８３	ＢＣ＝０	背景色は黒
＄Ｄ３８４	ＮＣ＝８０	桁数は８０
＄Ｄ３８５	ＮＬ＝２５	行数は２０
＄Ｄ３８６	ＳＬ＝０	全画面がスクロールモード画面
＄Ｄ３８７	ＮＳ＝２５	
＄Ｄ３８８	ＦＤ＝０	ファンクションキーは表示しない
＄Ｄ３８９	ＥＲＳ＝１	初期設定後に画面消去する
＄Ｄ３８Ａ	ＧＲ＝０	カラー表示

ワークエリア．．ＤＭＡＰ（＝Display Map）について

＄Ｄ０３Ｄ～＄Ｄ０４４は，画面の構成を示すワークエリアで，スクロールモード画面・ページモード１画面・ページモード２画面・ＰＦキー表示画面の開始行と終了行が２バイトずつで，計８バイトあります．設定はＩＮＩＴルーチンで行っています．これらの画面のうち，設定されなかった画面は，その開始行と終了行が等しくなっています．このワークエリア．．ＤＭＡＰは，ＰＵＴ，ＧＥＴコマンドやＥＲＡＳＥコマンド内において参照される重要な定数となります（終了行は，実際には，その画面が終了した行の次の行になっています）．

図3・3・1 ..DMAP



HOME

HOME	＄E1DA～E1F9
CHGWIN	＄E1F2

機　能　現在のウィンドウ画面を決定し，関連するワークエリアを設定します．

レジスタ　A，B，X，Y，U

ワークエリア	.BPY	＄D045～＄D04C	(.BPY～.BVAD)
	.MDRG	＄D04D～＄D04F	ウィンドウ画面情報
	GTTOPC	＄D050～＄D057	(GTTOPC～GTBTMA) ウィンドウ画面範囲
	.CY	＄D058～＄D05F	(.CY～.CVAD)
サブルーチン	BFPTRL	＄E1FA	バッファポインタのアドレスを求める
	GETRNG	＄E975	ウィンドウのアドレスを求める
	CURSET	＄EDE8	カーソルの位置を設定する
	INDICT	＄EE0E	PFキーの表示

解　説　次の動作を順番に行っていきます．

1）PFキー（ファンクションキー）の表示を行う（INDICT)．ただし表示するように設定されている場合に限る．

2）現在のウィンドウ画面を決定する．

3）決定されたウィンドウのキャラクタバッファおよびアトリビュートバッファ内

の範囲（バッファアドレス）を求める（GETRNG）．

4）現在のウィンドウ画面の先頭にバッファポインタを設定し，そのポインタが指す位置のバッファ内のアドレス(バッファアドレス)を求める(BFPTRL)．

5）バッファポインタに基づいて，カーソルの位置を決定する（CURSET）．

CHGWINでは，この動作のうち1)と2)の動作は行いません．よって，ワークエリアについては，．BPY（$D045）と，．MDRG（$D04D～$D04F）は設定しません．つまり，ウィンドウ画面の決定をするのではなく，現在のウィンドウ画面に付随するワークエリアの初期化だけを行うのです．

現在のウィンドウ画面というのは，スクロール，ページ1，ページ2の各画面のうちで，現在選択されている画面のことを指し，その選択方法（HOMEにおいて）は，..DMAP（$D03D～$D044）をスクロール画面，ページ1画面，ページ2画面の順に検索していき，その画面の開始行と終了行が等しくないものが表われたとき，つまりその画面が存在するときは，その画面を現在のウィンドウ画面として設定します．ウィンドウ画面についての情報は，．MDRG（$D04D～$D04F）に格納されます．

.MDRG	$D04D	画面(選択されたウィンドウ画面)の開始行
.MDRG＋1	$D04E	画面の終了行
.MDRG＋2	$D04F	画面の番号(スクロール…1，ページ1…2，ページ2…3)

．BPY，．BPX（および，．CY，．CX）は，ウィンドウ画面の先頭の位置（開始の左端）に設定され，．BPADなどのワークエリアは，その位置に対するバッファ・VRAM上のアドレスが入ります．

BUFPTR

BFPTRL	$E1FA ～ $E1FF
BUFPTR	$E1FC

機　能　現在の．BPX，．BPY（バッファポインタのX座標とY座標）から，バッファポインタのキャラクタバッファ上のアドレス(．BPAD)，アトリビュートバッファ上のアドレス(．BAAD)，および，VRAM上のアドレス(．BVAD)を求めます．

レジスタ　A，B，X，Y

ワークエリア　.ＢＰＹ　　＄Ｄ０４７　バッファポインタのＹ座標
.ＢＰＸ　　＄Ｄ０４８　バッファポインタのＸ座標
⇩
.ＢＰＡＤ　＄Ｄ０４９～＄Ｄ０４Ａ　バッファアドレス
.ＢＡＡＤ　＄Ｄ０４Ｂ～＄Ｄ０４Ｃ　バッファアドレス
.ＢＶＡＤ　＄Ｄ０４Ｄ～＄Ｄ０４Ｅ　ＶＲＡＭ上のアドレス

サブルーチン　ＢＢＡＤＲ　＄Ｅ２００　バッファ上のアドレスを求める
ＢＶＡＤＲ　＄Ｅ２１Ｃ　ＶＲＡＭ上のアドレスを求める

解　説　ＢＦＰＴＲＬがエントリのときは，.ＢＰＸが０に設定されるため，.ＢＰＹが示す行の先頭（左端）のアドレスが求まります．

ＶＲＡＭ上のアドレスは，青の画面についてのみ求めます．他の画面は，それに＄４０００，＄８０００のオフセットをかけてやればよいのです．

> ＶＲＡＭ上のアドレスについて
>
> ＶＲＡＭ上のアドレスというのは，基本的には青の画面（＄００００～＄３ＦＦＦ）のＶＲＡＭのアドレスを指します．赤と緑の画面の同じ座標に対応するＶＲＡＭのアドレスは，上で述べたように，青の画面のアドレスに＄４０００，＄８０００のオフセットをかけてやれば求まります．
>
> なお，３－６節では，この青の画面を指すＶＲＡＭ上のアドレスのことを，グラフィックアドレスと呼んでいます．

ＢＡＤＲ

ＢＢＡＤＲ	＄Ｅ２００	～	＄Ｅ２０７
ＢＡＤＲＬ	＄Ｅ２０８	～	＄Ｅ２１Ｂ
ＢＡＤＲＮ	＄Ｅ２０Ａ		
ＢＡＤＲ	＄Ｅ２０Ｃ		

機　能　キャラクタのＸ座標とＹ座標から，その座標のキャラクタバッファアドレスとアトリビュートバッファアドレスを求めます．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ

入力情報　解説参照

出力情報　Xレジスタ　キャラクタバッファアドレス

Yレジスタ　アトリビュートバッファアドレス

.ＢＰＡＤ　Xレジスタと同じ内容（ＢＢＡＤＲのみ）

.ＢＡＡＤ　Yレジスタと同じ内容（ＢＢＡＤＲのみ）

ワークエリア　..ＬＮＣ　＄Ｄ０３６　１行の桁数

.ＢＰＹ　＄Ｄ０４５　（ＢＢＡＤＲ，ＢＡＤＲＮのみ）

.ＢＰＸ　＄Ｄ０４６　（ＢＢＡＤＲ，ＢＡＤＲＮのみ）

.ＢＰＡＤ　＄Ｄ０４７～＄Ｄ０４８　（ＢＢＡＤＲのみ）

.ＢＡＡＤ　＄Ｄ０４９～＄Ｄ０４Ａ　（ＢＢＡＤＲのみ）

解　説　エントリの違いによって，求める座標のアドレスと，入力情報が次のように変わります．

１）ＢＡＤＲ　　ＡレジスタY座標，ＢレジスタがX座標を示す座標のアドレス

２）ＢＡＤＲＬ　ＡレジスタがY座標，X座標０のアドレス

３）ＢＡＤＲＮ　．ＢＰＹがY座標，．ＢＰＸがX座標を示す座標のアドレス

４）ＢＢＡＤＲ　ＢＡＤＲＮと同じ

アドレスの計算方法は，次のようになります．

キャラクタバッファ

　Xレジスタ＝（Y座標）×..ＬＮＣ＋（X座標値）＋＄Ｃ０００

アトリビュートバッファ

　Yレジスタ＝Xレジスタの値＋＄０７Ｄ０

※..ＬＮＣには，１行の桁数が入っています．

　次の例は水平位置２５，垂直位置１２の座標のバッファアドレスをX，Yレジスタに求めます．

```
LDA  #25
LDB  #12
JSR  BADR
```

キャラクタバッファとアトリビュートバッファについて

ユーザに見えているキャラクタはＶＲＡＭ上に書かれたキャラクタのフォントであり，キャラクタ単位のＸ，Ｙ座標にどのキャラクタが書かれているか，また，そのキャラクタの色は何かなどという情報を記憶しているのが，キャラクタバッファ（文字コードバッファ）とアトリビュートバッファです．

キャラクタバッファは，その座標に書かれているキャラクタのアスキーコードを記憶しています．通常はオーダ動作をするため，＄００～＄１Ｆ，＄７Ｆの値をとることはありませんが，コンソール動作の制御をしてオーダ動作を禁止した場合は，＄００～＄ＦＦの値をとります．

アトリビュートバッファは，その座標のアトリビュートの属性を示すコード（アトリビュート文字）を示します．コードは１バイトで，そのビットパターンは次のとおりです．

図3・3・2　アトリビュートコード

ビット7	6	5	4	3	2	1	0
F	M	／	P	R	CL		

ＣＬ　文字の表示色（０～７）
Ｒ　１のとき反転表示
Ｐ　０のとき非保護
　　１のとき保護
Ｍ　１のときオペレータ
　　によって変更された領域
Ｆ　１のときフィールドの先頭

このキャラクタバッファ（文字コードバッファ）とアトリビュートバッファのアスキーコードとアトリビュートコードによって，サブシステムは画面の１文字を認識しています．この２つのバッファは，総称してコンソールバッファと呼ばれ，各々次の領域にあります．

キャラクタバッファ　　：　＄Ｃ０００～＄Ｃ７ＣＦ
アトリビュートバッファ　：　＄Ｃ７Ｄ０～＄ＣＦ９Ｆ（＄ＣＦＡ０）

画面と各々のバッファとの対応を図３・３・３と図３・３・４に示します．１行の桁数が変わると，構成が変わるので注意して下さい．また画面の行数が２０行のときは，Ｙ座標が１９のところまでが使用されます．

水平位置と垂直位置について

グラフィックのＸ座標とＹ座標を区別するために，キャラクタ単位のＸ座標とＹ座標のことを水平位置，垂直位置と呼ぶことがあります．

図3・3・3　キャラクタバッファとアトリビュートバッファ

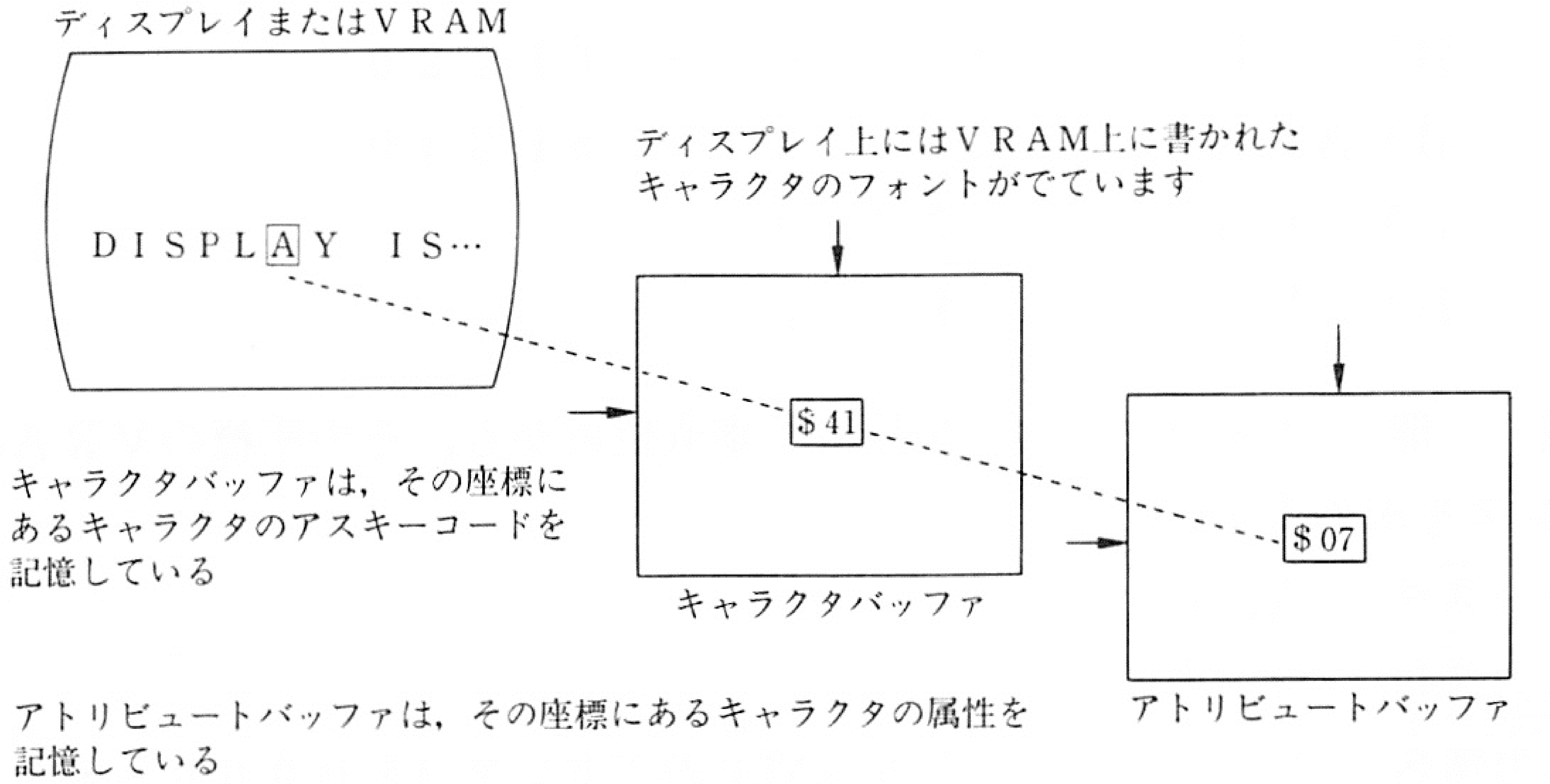


図3・3・4　各バッファの座標とアドレスとの対応

a．Width40のとき

キャラクタバッファ

	X＝0	X＝1	………	X＝38	X＝39
Y＝0	C000	C001	……	C026	C027
Y＝1	C028	C029	……	C04E	C04F
Y＝2	C050	C051	……	C076	C077
Y＝3	C078	C079	……	⋮	⋮
⋮	⋮	⋮	……	C2F6	C2F7
Y＝19	C2F8	C2F9	……	C31E	C31F
⋮	⋮	⋮	……	⋮	⋮
Y＝24	C3C0	C3C1	……	C3E6	C3E7

アトリビュートバッファ

	X＝0	X＝1	………	X＝38	X＝39
Y＝0	C7D0	C7D1	……	C7F6	C7F7
Y＝1	C7F8	C7F9	……	C81E	C81F
Y＝2	C820	C821	……	C846	C847
Y＝3	C848	C849	……	⋮	⋮
⋮	⋮	⋮	……	CAC6	CAC7
Y＝19	CAC8	CAC9	……	CAEE	CAEF
⋮	⋮	⋮	……	⋮	⋮
Y＝24	CB90	CB91	……	CBB6	CBB7

b．Width80のとき

キャラクタバッファ

	X＝0	X＝1	………	X＝78	X＝79
Y＝0	C000	C001	……	C04E	C04F
Y＝1	C050	C051	……	C09E	C09F
Y＝2	C0A0	C0A1	……	C0EE	C0EF
Y＝3	C0F0	C0F1	……	⋮	⋮
⋮	⋮	⋮	……	C5EE	C5EF
Y＝19	C5F0	C5F1	……	C63E	C63F
⋮	⋮	⋮	……	⋮	⋮
Y＝24	C780	C781	……	C7CE	C7CF

アトリビュートバッファ

	X＝0	X＝1	………	X＝78	X＝79
Y＝0	C7D0	C7D1	……	C81E	C81F
Y＝1	C820	C821	……	C86E	C86F
Y＝2	C870	C871	……	C8BE	C8BF
	C8C0	C8C1	……	⋮	⋮
⋮	⋮	⋮	……	CDBE	CDBF
Y＝19	CDC0	CDC1	……	CE0E	CE0F
⋮	⋮	⋮	……	⋮	⋮
Y＝24	CF50	CF51	……	CF9E	CF9F

※Width２０のときは，Y＝１９の行までが用いられます

ＢＶＡＤ

ＢＶＡＤＲ	＄Ｅ２１Ｃ	～ ＄Ｅ２２０
ＢＶＡＤＬ	＄Ｅ２２１	～ ＄Ｅ２４０
ＢＶＡＤＮ	＄Ｅ２２３	
ＢＶＡＤ	＄Ｅ２２５	

機　能　キャラクタの水平位置と垂直位置から，その座標のＶＲＡＭ上のアドレスを求めます．

レジスタ　Ａ，Ｂ

入力情報　解説参照

出力情報

Ｄレジスタ	ＶＲＡＭ上のアドレス（＄００００～＄３ＦＦＦ）
．ＢＶＡＤ	Ｄレジスタと同じ内容（ＢＶＡＤＲのみ）

ワークエリア

．．ＶＣＵ	＄Ｄ０３９～＄Ｄ０３Ａ（実際は＄Ｄ０３Ａのみ）
．．ＶＬＵ	＄Ｄ０３Ｂ～＄Ｄ０３Ｃ
．ＢＰＹ	＄Ｄ０４５　（ＢＶＡＤＲ，ＢＶＡＤＮのみ）
．ＢＰＸ	＄Ｄ０４６　（ＢＶＡＤＲ，ＢＶＡＤＮのみ）
．ＢＶＡＤ	＄Ｄ０４Ｂ～＄Ｄ０４Ｃ　（ＢＶＡＤＲのみ）

解　説　エントリの違いによって，入力情報と求めるアドレスが次のようにかわります．

１）ＢＶＡＤ　Ａレジスタが垂直位置，Ｂレジスタが水平位置を示す座標のアドレス

２）ＢＶＡＤＬ　Ａレジスタが垂直位置，水平位置０の座標のアドレス

３）ＢＶＡＤＮ　．ＢＰＹが垂直位置，．ＢＰＸが水平位置を示す座標のアドレス

４）ＢＶＡＤＲ　ＢＶＡＤＮと同じ

アドレスの計算方法は，次のようになります

Ｄレジスタ＝（水平位置）×．．ＶＣＵ＋（垂直位置）×．．ＶＬＵ

（水平位置）×．．ＶＣＵ：Ｘ座標によるアドレスの計算
（垂直位置）×．．ＶＬＵ：Ｙ座標によるアドレスの計算

※．．ＶＣＵ，．．ＶＬＵには，各々，水平位置，垂直位置が１つの大きくなったとき，ＶＲＡＭ上のアドレスをどれだけ大きくするかという，ＶＲＡＭ上の桁単位とＶＲＡＭ上の行単位が入っています．

次の例は，第１２行の先頭の座標のＶＲＡＭ上のアドレスを，Ｄレジスタに求めるものです．

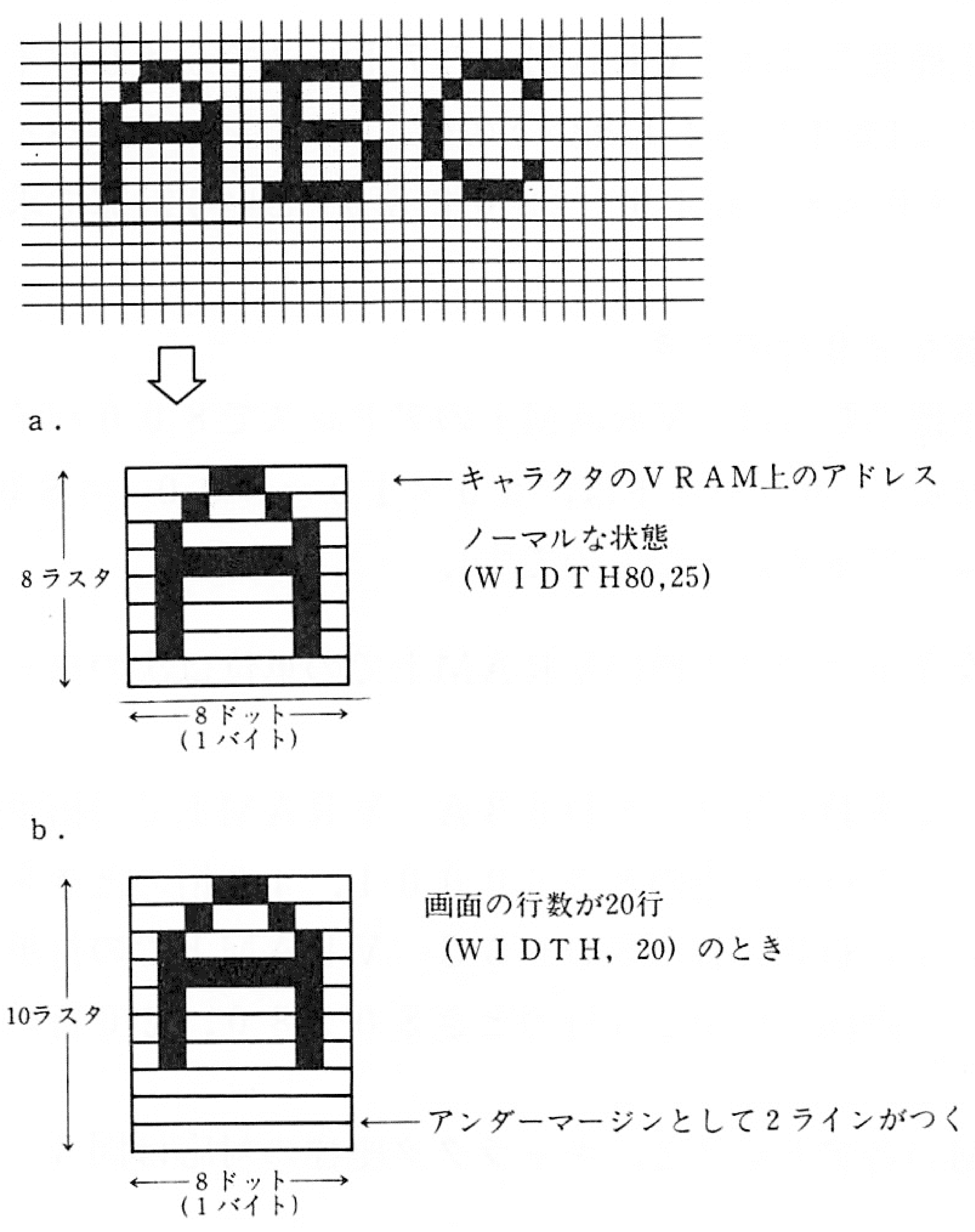
```
LDA  #12
JSR  BVADL
```

ＶＲＡＭ上のキャラクタ座標について

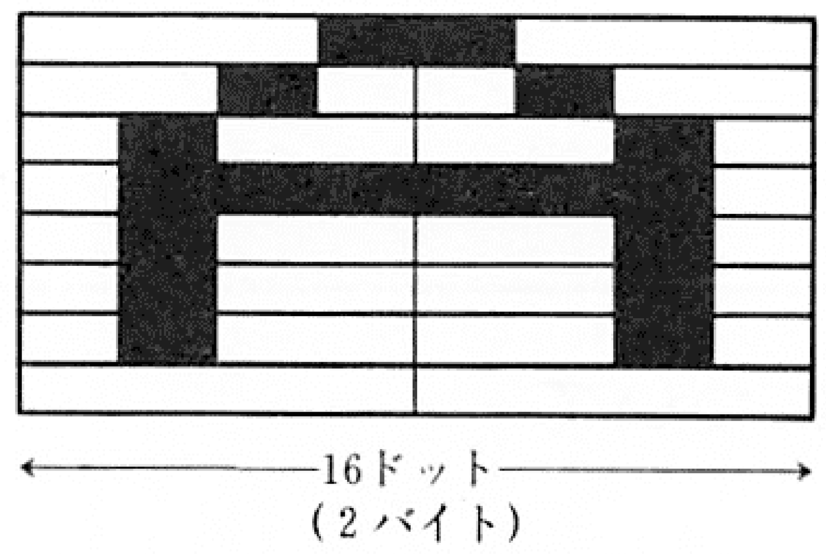
キャラクタのフォントは，横８ドット，縦８ラスタで構成されています（ラスタというのは，縦方向のドットのことです）．だから，キャラクタ単位で横の１座標というのはグラフィックでの８ドット（１バイト）となり，縦の１座標というのは，８ラスタになります（図３・３・５のａ）．

画面の行数が２０行のときは，フォントの下にアンダーマージン（縦方向を調整するための空ライン）として２ラインがつきますので，縦の１座標は１０ラスタになります（図３・３・５のｂ）．また，１行の桁数が４０桁のときは，フォントが横２倍に拡張されるので，横の１座標は１６ドット（２バイト）になります（図３・３・５のｃ）．

図3・3・5　キャラクタのフォント



c．



1行の桁数が40桁
(WIDTH40）のとき

※図は実際の縦横の比率と一致していません

以上のことから，キャラクタ座標は次のようにアドレスと対応します．

1桁の桁数が80桁のとき

横に1座標進むには，VRAM上のアドレスで1バイト（8ドット＝1バイト）進まなければなりません．

1行の桁数が40桁のとき

横に1座標進むには，VRAM上のアドレスで2バイト進まなければなりません．

画面の行数が25行のとき

縦に1座標進むには，VRAM上のアドレスで640バイト進まなければなりません(640バイトというのは，1ラインが80バイトで構成されますから，80×8＝640（＝＄0280）という計算で求められたものです)．

画面の行数が20行のとき

縦に1座標進むには，VRAM上のアドレスで800バイト進まなければなりません(800バイトは，80×10＝800（＝＄0320）という計算によって求められたものです)．

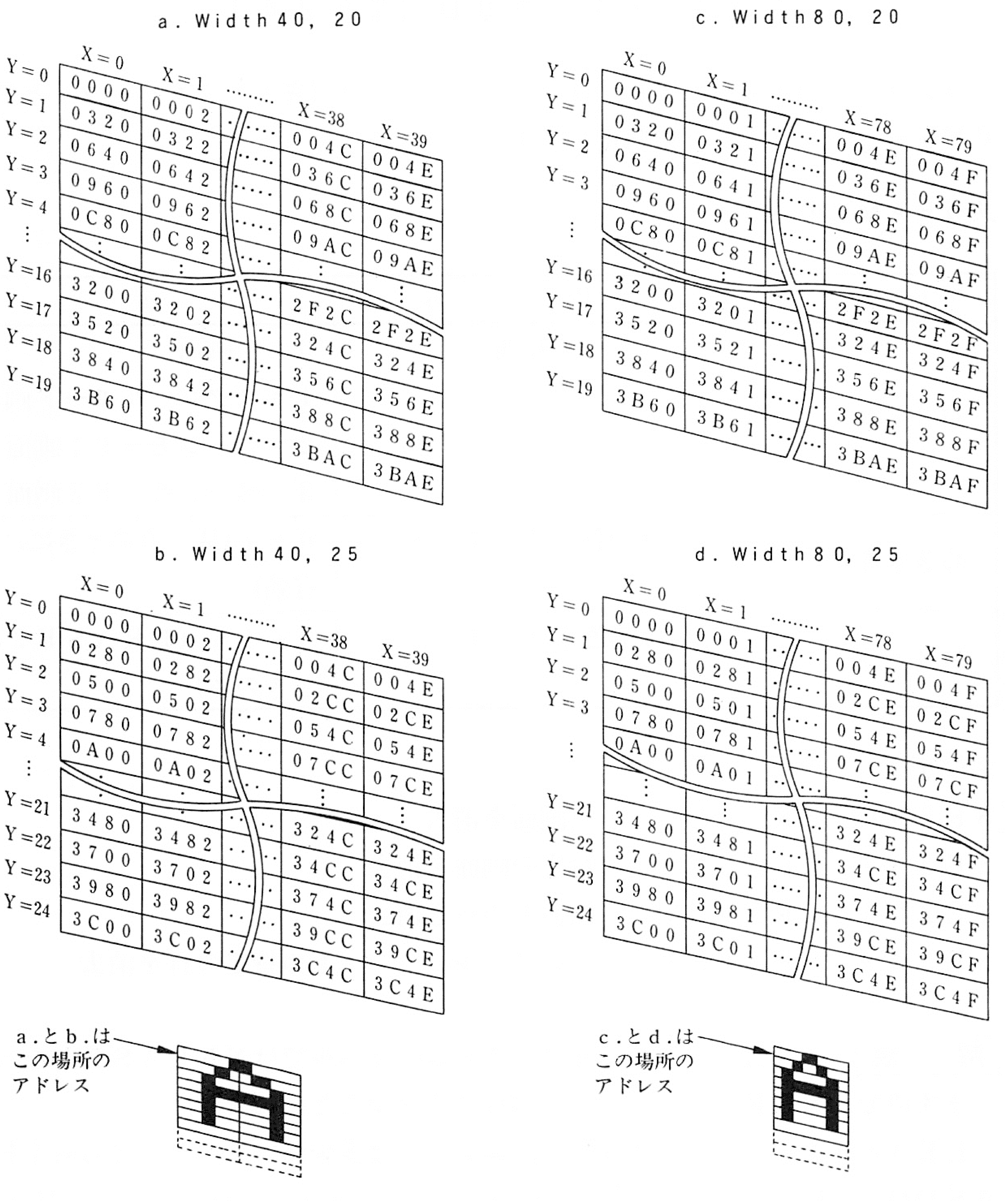
このようなキャラクタ座標のVRAM上での単位は次のワークエリアが持っています．

..VCU：＄D039～＄D03A　VRAM上での桁単位
　1行80桁のとき＄0001，40桁のとき＄0002

..VLU：＄D03B～＄D03C　VRAM上での行単位
　画面の行が25行のとき＄0280，20行のとき＄0320

VRAMの各アドレスと，キャラクタ座標の対応は図3・3・6を見て下さい．

図3・3・6　キャラクタのＶＲＡＭ上のアドレスと座標との対応



3－3－2 ERASE　$02（$E24B）
ERASE2　$0D（$E241）

コンソール画面をクリアし，コンソールバッファ（キャラクタバッファとアトリビュートバッファ）を初期化します．

共有RAM上のパラメータ

アドレス	記　号	内　　容	値
D383	W	イレーズ範囲	0＝全画面 1＝スクロールモード画面 2＝ページモード1画面 3＝ページモード2画面
D384	BC	背景色カラーコード	0～7（W＝0のときにのみ有効）
D385	FC	文字色カラーコード	0～15（ERASE2のときのみ）

主なサブルーチン

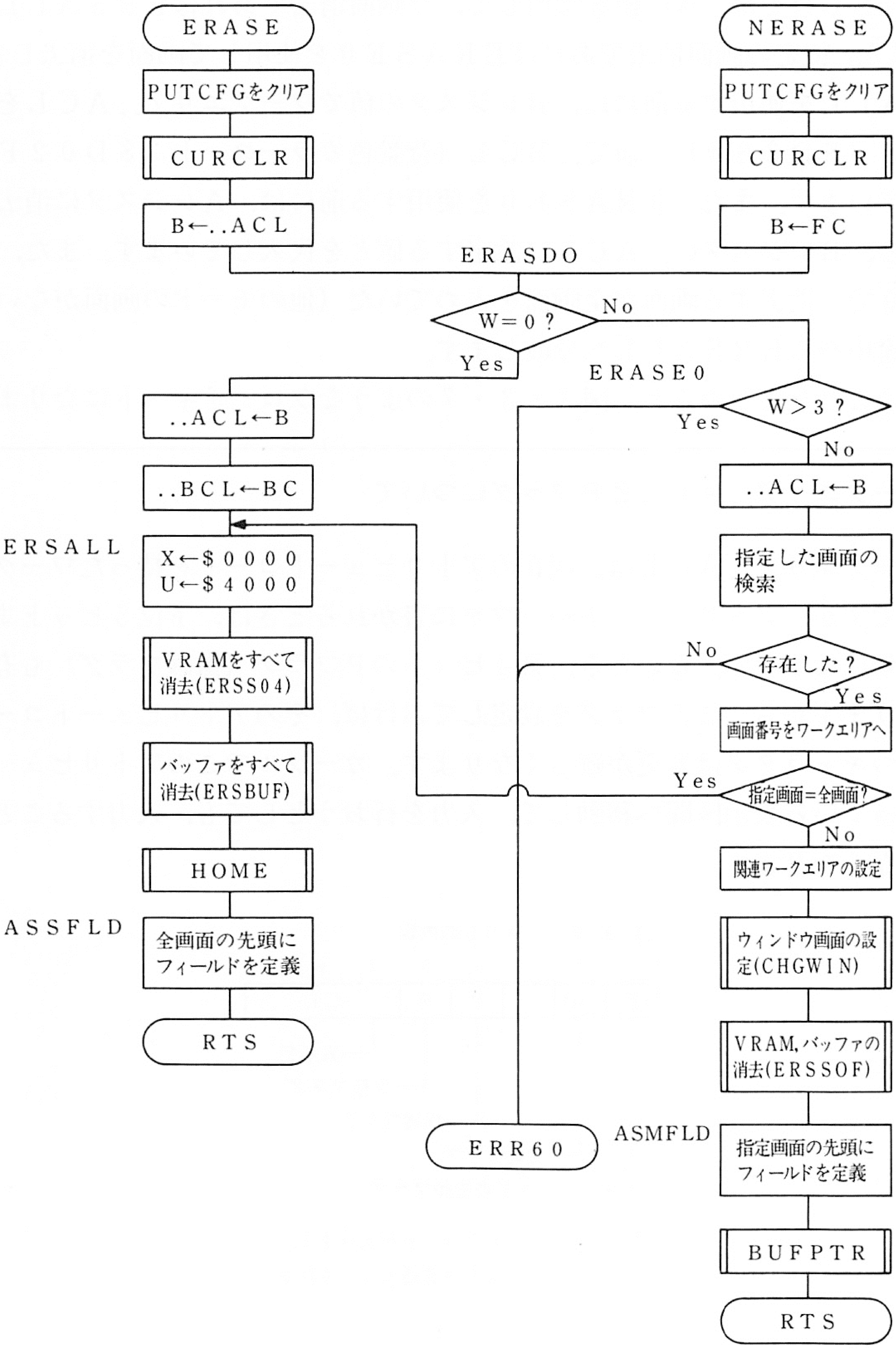
$E25E　ERSALL　全画面を消去
$E298　ERASE0　特定の画面を消去
$E2D3　ERSVRM　画面上の行を消去
$E331　ERSBUF　コンソールバッファ上の特定の行を消去

解　説　ERASE2は，サブシステムのラベルではNERASEとなっていますので，NERASEというラベル名で呼ぶことにします．

ERASEおよびNERASEは，エントリは多少違いますが，どちらもERASDO（$E252）というラベルのイレーズ（消去）実行ルーチンにおいて，画面をイレーズしています．ERASEとNERASEの相違は，ERASDOルーチンに入る前に，Bレジスタに何が設定されるかという点にあります．ERASDOでは，Bレジスタの値に基づいて，ワークエリア..ACL（アトリビュートカラーコード）を設定しています．ERASEとNERASEでは，次のように違います．

ERASE　現在の..ACL（$D02E）の値でBレジスタを設定
NERASE　FC（$D385）の値でBレジスタを設定

図3・3・7　ＥＲＡＳＥおよびＮＥＲＡＳＥのフローチャート



よって，ＮＥＲＡＳＥでは，イレーズ後の文字色が変更可能なのです．

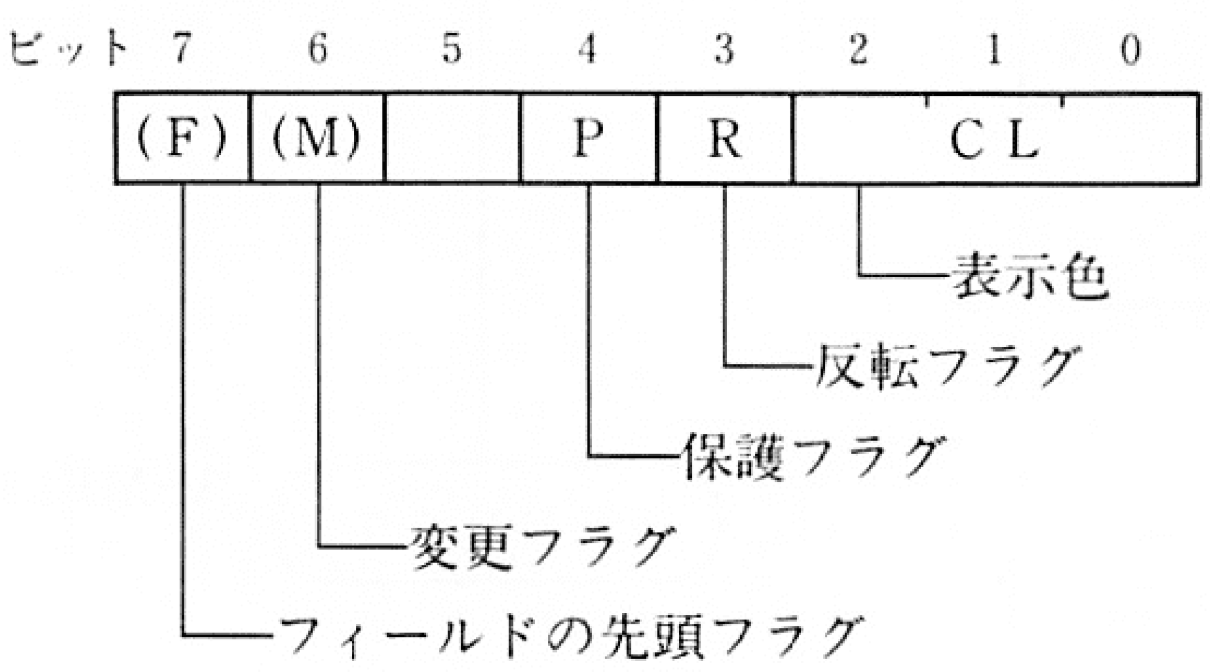
ＥＲＡＳＤＯでは，Ｗの値を参照して，全画面消去であればＥＲＳＡＬＬを使用し，また，特定の画面消去であればＥＲＡＳＥ０を使用して画面を消去します．ＥＲＳＡＬＬを使用する前には，Ｂレジスタの値でワークエリア..ＡＣＬを設定し，ＢＣ（＄Ｄ３８４）の値で..ＢＣＬ（背景色カラーコード：＄Ｄ０２Ｆ）を設定しています．また，ＥＲＡＳＥ０を使用する前には，Ａレジスタに消去画面の番号と，Ｂレジスタに..ＡＣＬに設定する値とを代入しています．また，ＥＲＡＳＥ０で，消去する画面が全画面を占めていた（他のモードの画面がない）場合は，途中からＥＲＳＡＬＬへ分岐します．

以上のことをまとめると，図３・３・７のようなフローチャートになります．

ワークエリア..ＡＣＬとＰフラグについて

ワークエリア..ＡＣＬは，現在のアトリビュートコードを持ったワークエリアですが，アトリビュートバッファに書かれるときは，下位５ビットまでが有効になります．このとき，第４ビットのＰフラグ（保護フラグ）も有効になりますから，このフラグを設定しておけば，そのアトリビュートコードを持つキャラクタは変更が難しくなります．カーソルでこのアトリビュートを持っている表示区間へ移動して，入力を行おうとしても，入力することはできなくなります．

図3・3・8　..ＡＣＬの内容



現在のアトリビュートコードが入ります．
ただし，ビット７と６は普通セットされません．

この..ＡＣＬは，ＥＲＡＳＥ２コマンドのＦＣやＰＵＴコマンドのＳＦ(Start Field）オーダで設定できます（ＦＣは，０～１５までとありますが，実際は０～２５５までの値を，..ＡＣＬに代入することができます．このとき，ＦフラグやＰフラグを立てておくと，画面上でキー入力ができなくなります)．例えば，Ｆ－ＢＡＳＩＣでは，＄０１Ｅ５番地の内容をサブシステムに出力

し，それが..ＡＣＬの値となるため，次のようなプログラムで，表示部分の変更を禁止することができます．

```
10 POKE&H01E5,&H17:PRINT"This is the protected field."
20 POKE&H01E5,&H12:PRINT"You can't erase this field without using CLS or EL key."
30 POKE&H01E5,&H07
```

パレットレジスタ（＄ＦＤ３８）について

直接サブシステムとは関係ありませんが，メイン側のパレットレジスタについての特殊な使用法について説明します．

ＦＭ－７では，パレットレジスタとして＄ＦＤ３８～＄ＦＤ３Ｆが用意されていますが，＄ＦＤ３８は特殊なパレットレジスタで，ここを書き換えることにより，ボーダカラー（画面の境界の外の色）を変化させることができます．実際に有効なのは下位３ビットだけです．Ｆ－ＢＡＳＩＣでは，ＣＯＬＯＲ文でパレットコードを定義するときでも，＄ＦＤ３８に対しては常に０を出力しているようにしていますので，下のプログラムのように直接書き換えなければなりません．

```
10 FOR I= 0 TO 255 : POKE&HFD38,I : NEXT : GOTO 10
```

ＥＲＳＡＬＬ

ＥＲＳＡＬＬ　　$Ｅ２５Ｅ　～　$Ｅ２９７
ＡＳＳＦＬＤ　　$Ｅ２７Ｄ

機　能	全画面をクリアします．各画面(スクロール，ページ１，ページ２)の先頭にフィールドの始まりを定義します．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	ＣＳＬＣＤＳ	$Ｄ０２３	単色表示フラグ
	..ＤＭＡＰ	$Ｄ０３Ｄ～$Ｄ０４４	画面情報
	.ＣＶＴＯＰ	$Ｄ０７Ｄ～$Ｄ０７Ｅ	消去行情報
サブルーチン	ＨＯＭＥ	$Ｅ１ＤＡ	ウィンドウ画面の決定
	ＢＡＤＲＬ	$Ｅ２０８	バッファのアドレスを求める
	ＥＲＳＳ０４	$Ｅ２Ｆ４	ＥＲＳＶＲＭの途中のエントリ
	ＥＲＳＢＵＦ	$Ｅ３３１	バッファの特定の行を消去する

解　説　動作は，ＶＲＡＭ消去とバッファ消去，そして各画面のフィールド設定の，３つの動作にまとめることができます．

１）ＶＲＡＭ消去

Ｘレジスタに$００００を，Ｕレジスタに$４０００を代入することで，全画面の消去であることを指定し，ＥＲＳＳ０４を使用してすべてのＶＲＡＭをクリアします．消去している間は，ＣＲＴ表示フラグ（$Ｄ４０８）をＯＦＦにして，画面表示を中止しています．また，単色表示モードのときでも他の画面（青の画面，赤の画面）も同時に消去できるよう，ＥＲＳＳ０４を呼びだしている間だけ，ワークエリアＣＳＬＣＤＳ（単色表示フラグ）を符号反転しています．

２）バッファ消去

ワークエリア..ＤＭＡＰ中から，ページ１画面の開始行とＰＦキー表示画面の終了行を取り出して，各々バッファ消去開始行およびバッファ消去終了行として，ワークエリア .ＣＶＴＯＰに設定して，ＥＲＳＢＵＦを使用してコンソールバッファを消去します．そして，ＨＯＭＥを呼んで現在のウィンドウ画面を設定します．

３）フィールドの設定

スクロール，ページ１，ページ２，ＰＦキー表示の各画面の先頭の位置に，フィールドの始まりを設定します（その座標のアトリビュートコードのビット７を１にして，Ｆ（フィールドの先頭）フラグを立てています）．また，アトリビュート

バッファの終了番地の次の番地に対して，ビット７を１にしています．この理由は後で述べます．ＡＳＳＦＬＤがエントリの場合は，この３)の処理だけを行います．

ＥＲＡＳＥ０

ＥＲＡＳＥ０　　$E298 ～ $E2D2
ＡＳＭＦＬＤ　　$E2C5

機　能	特定の画面をクリアし，その画面の先頭にフィールドの始まりを定義します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	Ａレジスタ	消去する画面の番号（１～３）	
	Ｂレジスタ	アトリビュートコード	
ワークエリア	..ＡＣＬ	$D02E	現在のアトリビュートコード
	..ＤＭＡＰ	$D03D～$D044	画面情報
	.ＢＰＹ	$D045	バッファポインタのＹ座標
	.ＭＤＲＧ	$D04D～$D04F	ウィンドウ画面情報
	.ＣＶＴＯＰ	$D07D～$D07E	消去行情報
サブルーチン	ＣＨＧＷＩＮ	$E1F2	ウィンドウ画面に関する情報の設定
	ＢＵＦＰＴＲ	$E1FC	バッファポインタのアドレスを求める
	ＢＡＤＲＬ	$E208	指定した行のバッファアドレスを求める
	ＥＲＳＡＬＬ	$E25E	全画面をクリアする
	ＥＲＳＳＯＦ	$E325	指定した行を消去する

解　説　動作は，画面の検索とウィンドウ画面の設定および消去とフィールドの設定という３つの動作にまとめることができます．

１）画面の検索

Ｂレジスタの値をワークエリア..ＡＣＬに設定した後，Ａレジスタで指定した画面が存在するかどうかを，ワークエリア..ＤＭＡＰを検索しながらチェックしていきます．もし，画面の番号が３より大きかったり，指定した画面が存在しない（画面の開始行と終了行が同じ値であった）場合には，ＥＲＲ６０（$E0CF）に分岐しています．指定した画面があれば，その画面の番号を，.ＭＤＲＧの３バイト目（$D04F）に格納しておきます．また，指定した画面が全画面で

あれば（他の画面がなければ）ＥＲＳＡＬＬに分岐して，そこで全画面を消去します．

２）ウィンドウ画面の設定および消去

指定した画面の開始行と終了行を．ＣＶＴＯＰと．ＭＤＲＧに各々設定し，開始行のみを．ＢＰＹに設定します．この後ＣＨＧＷＩＮでウィンドウ画面の情報を設定し，ＥＲＳＳＯＦでウィンドウ画面の部分のコンソールバッファとＶＲＡＭを消去します．

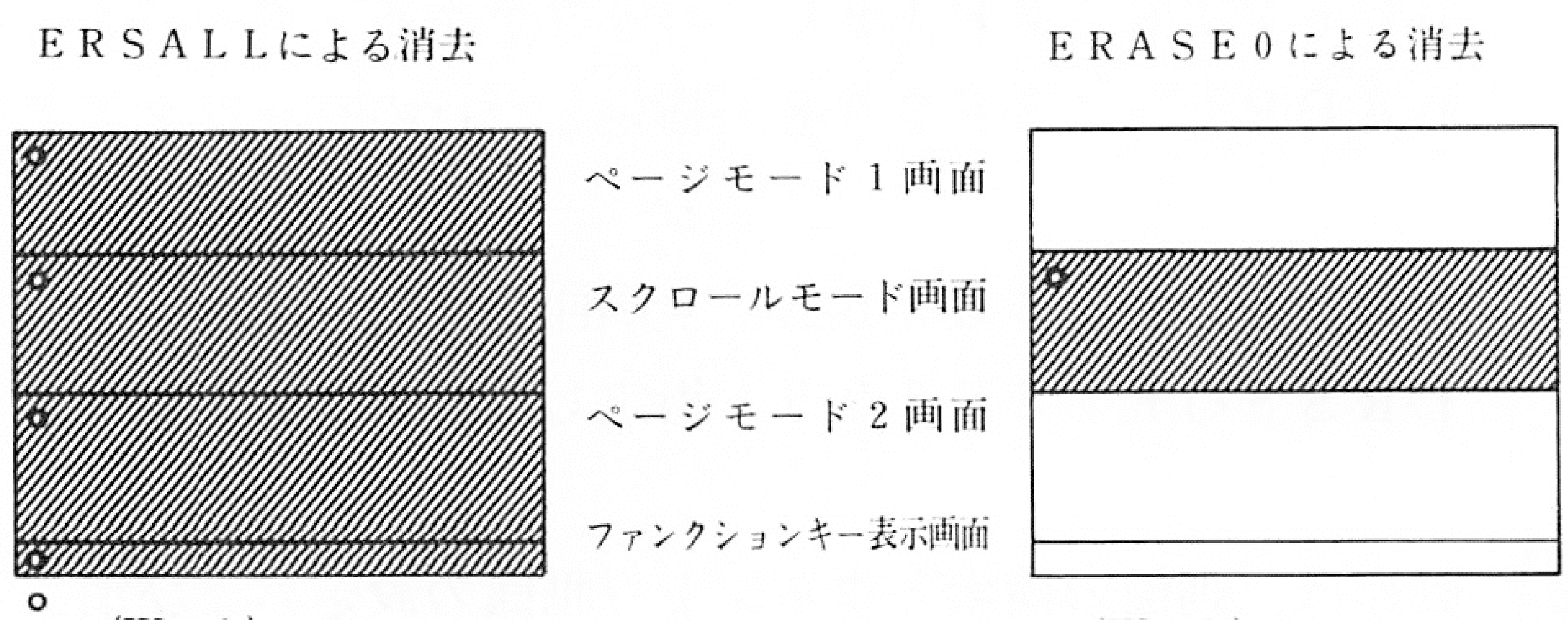
３）フィールドの設定

現在のウィンドウ画面の開始行（．ＭＤＲＧ）から，その先頭のアトリビュートバッファのアドレスを求め(ＢＡＤＲＬ)，そこに書かれているアトリビュートコードのＦフラグを立てて，フィールドの始まりであることを定義します．最後に，バッファポインタの示す座標のアドレス（バッファアドレス：この場合は．ＢＰＸが０になっているので，．ＢＰＹの示す行の先頭のアドレス）を，ＢＵＦＰＴＲを使用して，ワークエリアに設定します．

次の例は，ページモード１画面をクリアし，文字色を紫にします．

```
LDA   #2
LDB   #3
JSR   ERASE0
```

図3・3・9　ＥＲＳＡＬＬとＥＲＡＳＥ０による消去の違い



※斜線が消去部分
　○印がフィールドの始まりを定義する場所

ＥＲＳＶＲＭ

ＥＲＳＶＲＭ　　＄Ｅ２Ｄ３　～　＄Ｅ３２４
ＥＲＳＳ０４　　＄Ｅ２Ｆ４

| 機　能 | 画面上の特定の行を消去します．ただし，消去するのはＶＲＡＭだけで，コンソールバッファは消去しません． |

| レジスタ | Ａ，Ｂ，Ｘ，Ｙ，Ｕ |
| 入力情報 | ．ＣＶＴＯＰ　第１バイト目（＄Ｄ０７Ｄ）に消去開始行
第２バイト目（＄Ｄ０７Ｅ）に消去終了行の次の行 |

ワークエリア	ＣＳＬＣＤＳ	＄Ｄ０２３	単色表示フラグ
	．．ＡＣＬ	＄Ｄ０２Ｅ	現在のアトリビュートコード
	．．ＢＣＬ	＄Ｄ０２Ｆ	現在のバックカラー
	．ＣＶＴＯＰ	＄Ｄ０７Ｄ～＄Ｄ０７Ｅ	消去行情報
サブルーチン	ＢＶＡＤＬ	＄Ｅ２２１	指定した行の先頭のＶＲＡＭ上のアドレスを求める
	ＥＲＳＶ	＄Ｅ３６８	ＶＲＡＭの特定の領域を消去する

| 解　説 | 動作は．ＣＶＴＯＰ＋１（＄Ｄ０７Ｅ）とＣＳＬＣＤＳの値によって，次のように異ってきます． |

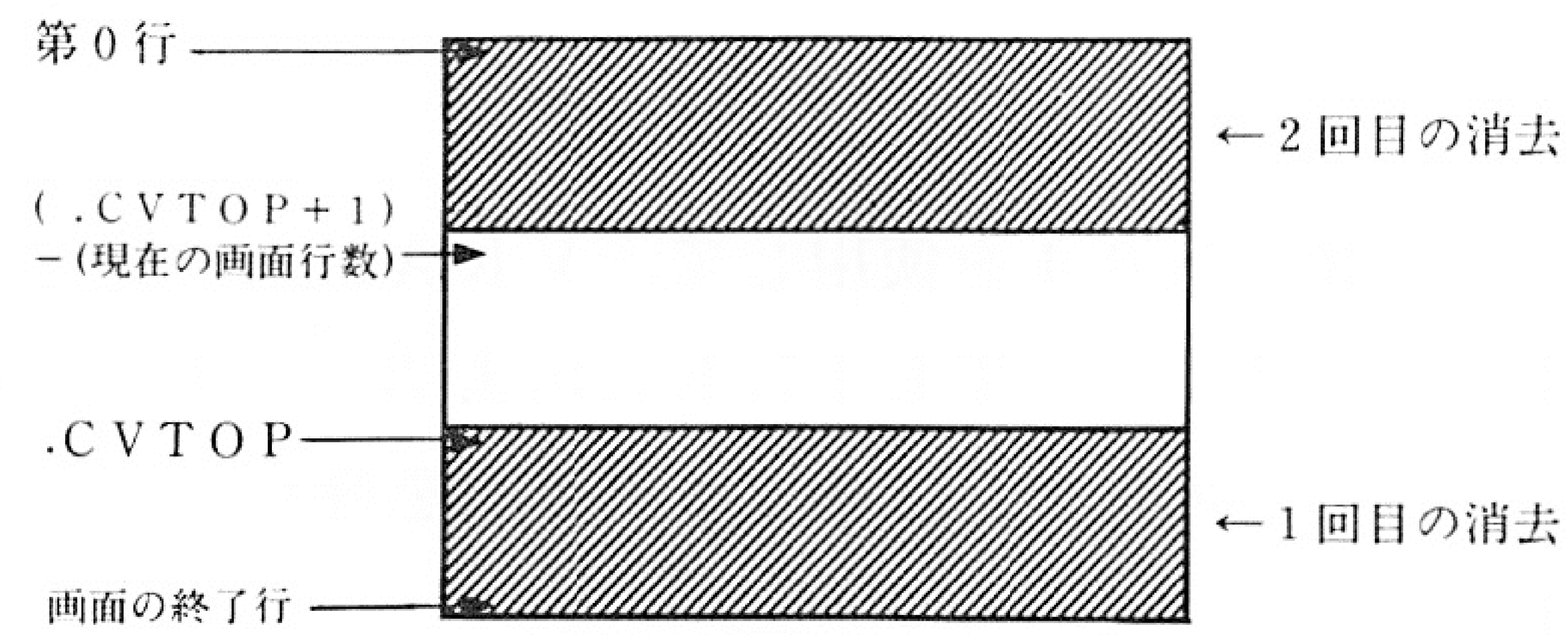
１）　．ＣＶＴＯＰ＋Ｉ（＝＄Ｄ０７Ｅ）

ｉ）　．ＣＶＴＯＰ＋１の値が，現在の画面の行数より小さいとき
画面の．ＣＶＴＯＰの示す行から，．ＣＶＴＯＰ＋１の示す行の手前の行までを消去します．

ii）　．ＣＶＴＯＰ＋１の値が，現在の画面の行数より大きいとき
画面の．ＣＶＴＯＰの示す行から最終の行までを消去し，次に画面の先頭行から，下の式で示される行の手前までを消去します．

（．ＣＶＴＯＰ＋１の値）－（現在の画面の行数）

図3・3・10　ＥＲＳＶＲＭの２回の消去



2）ＣＳＬＣＤＳ

ｉ）ＣＳＬＣＤＳ＝０（カラーモード）のとき

normal 表示　..ＢＣＬの値（背景色）で消去

reverse 表示　..ＡＣＬの値（文字色）で消去

ii）ＣＳＬＣＤＳ≧＄８０（単色表示モード）のとき

normal 表示　単色表示の背景色（黒）で消去

reverse 表示　単色表示の文字色（緑）で消去

iii）ＣＳＬＣＤＳの値がそれ以外（単色表示モードだが，他の画面も同時に消去したいとき）

normal　表示　黒色で消去

reverse　表示　緑色で消去

normal 表示か reverse 表示かは，ワークエリア..ＡＣＬのビット３（Ｒフラグ）を見て判断しています．

このルーチンを使用するときには，.ＣＶＴＯＰ＋１の内容が，.ＣＶＴＯＰの値よりも大きくなければならないので注意して下さい．同じ値であったり，小さいときは暴走します．また，ＥＲＳＳ０４からエントリする場合は，次のレジスタを設定しておいて下さい．

Ｘレジスタ　ＶＲＡＭ上の消去する先頭アドレス（＄００００〜＄４０００）

Ｕレジスタ　ＶＲＡＭ上の消去する終了アドレス（＄００００〜＄４０００）

ただし，Ｘレジスタの値はＵレジスタの値よりも小さく，かつその差は偶数で６以上でなければいけません．

以上の動作の総括として，図３・３・11にフローチャートを示しておきます．

次の例は，ＷＩＤＴＨ２０のとき，１６〜１９行および０〜６行の間をクリアします（ＶＲＡＭのみ）．

```
LDA  #16
LDB  #27
STD  .CVTOP
JSR  ERSVRM
```

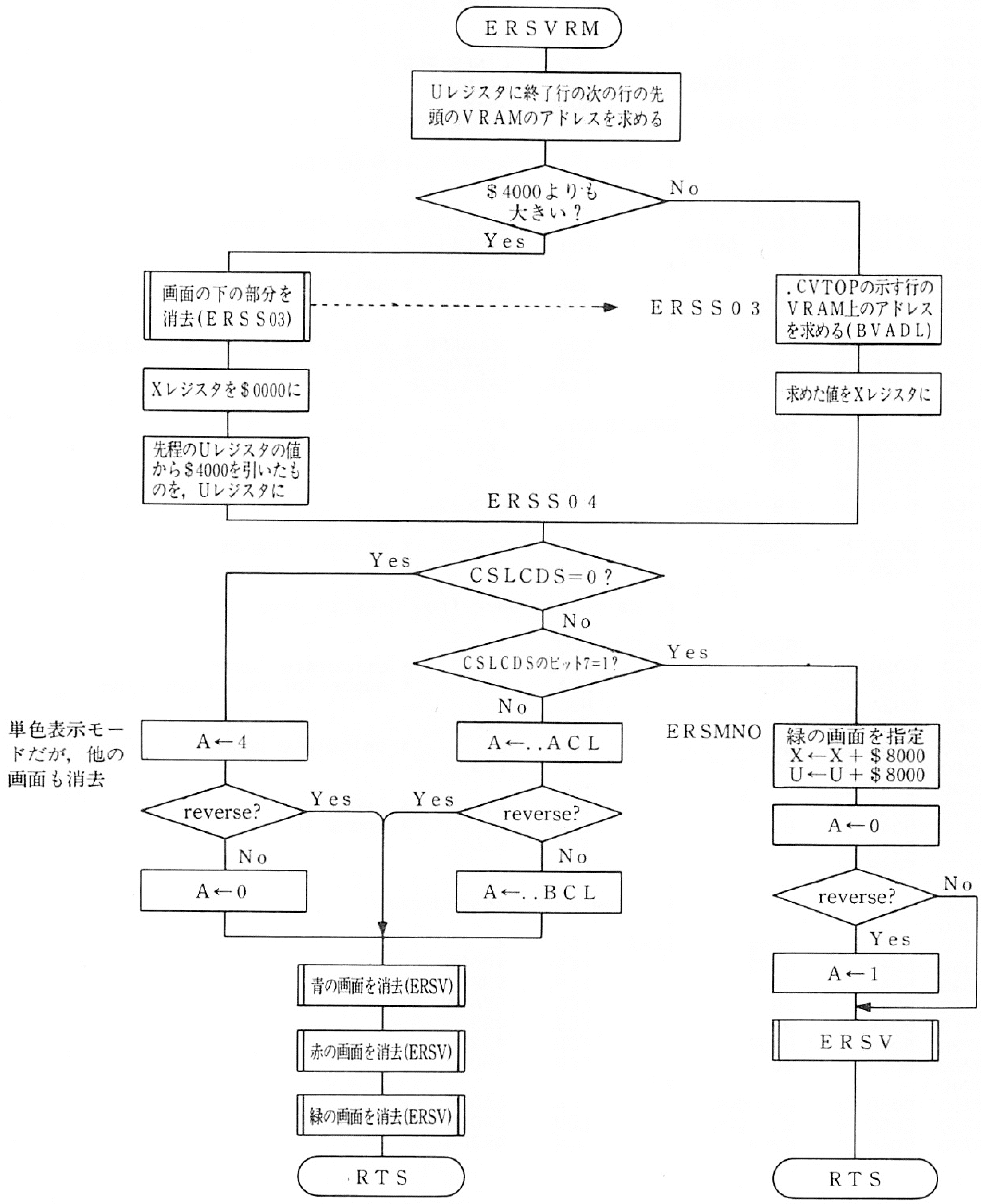
次のサンプルは，ＥＲＳＳ０４を使用して，ＶＲＡＭ上の特定のライン（０〜１９９）を消去するものです．ＳＴＬＩＮＥに消去開始ラインを入れ，ＬＩＮＥＳに消去するラインの数を入れて，このプログラムを呼びだすと，指定したラインがクリアされます．

```
10000                         ****************************************************
10010                         *
10020                         * erslin
10030                         *        erase lines
10040                         *
10050                         *        stline .. erase start line
10060                         *        lines  .. number of erase line
10070                         *
10080                                NAM     ERSLIN
10090                                OPT     MEM,NOGEN
10100   5000                         ORG     $5000
10110                         *
10120              FD05       SBSOUT EQU     $FD05
10121              FC80       SHARED EQU     $FC80
10130                         *
10140              5000       ERSLIN EQU     *
10150                         *
10160                         *  calculate address on VRAM
10170                         *
10180   5000 EC    8D 0064           LDD     STLINE,PCR
10190   5004 8D    30   5036         BSR     CALCU
10200   5006 ED    8D 005A           STD     SADDR,PCR
10210                         *
10220   500A 34    06                PSHS    D
10230   500C EC    8D 005A           LDD     LINES,PCR
10240   5010 8D    24   5036         BSR     CALCU
10250   5012 E3    E1                ADDD    ,S++
10260   5014 ED    8D 004E           STD     EADDR,PCR
10270                         *
10280                         *  run the program on shared ram
10290                         *
10300              5018       ERSLI1 EQU     *
10310   5018 B6    FD05              LDA     SBSOUT  * wait for ready
10320   501B 2B    FB   5018         BMI     ERSLI1
10330                         *
10340   501D 86    80                LDA     #$80    * halt sub system
10350   501F B7    FD05              STA     SBSOUT
10360                         *
10370   5022 CE    FC80              LDU     #SHARED * move program on shared ram
10380   5025 C6    1F                LDB     #ESERS-SERS
10390   5027 30    8D 001E           LEAX    SERS,PCR
10400                         *
10410              502B       ERSLI2 EQU     *
10420   502B A6    80                LDA     ,X+
10430   502D A7    C0                STA     ,U+
10440   502F 5A                      DECB
10450   5030 26    F9   502B         BNE     ERSLI2
10460                         *
10470   5032 7F    FD05              CLR     SBSOUT  * go the program
10480   5035 39                      RTS
10490                         *
10500                         *  calculate addr from Dreg to Dreg
10510                         *
10520              5036       CALCU  EQU     *
10530   5036 34    02                PSHS    A        * calculate lower
10540   5038 86    50                LDA     #80      * number of bytes per line
10550   503A 3D                      MUL
10560   503B 1F    01                TFR     D,X
10570   503D 35    04                PULS    B        * calculate upper
10580   503F 86    50                LDA     #80
10590   5041 3D                      MUL
10600   5042 1E    89                EXG     A,B      * D=B*256
10610   5044 30    8B                LEAX    D,X      * add D to X
10620   5046 1F    10                TFR     X,D
10630   5048 39                      RTS
10640                         *
10650                         *  program on shared ram
10660                         *
10670              5049       SERS   EQU     *
10680   5049       0000              FDB     $0000
10690   504B       3F                FCB     $3F
10700   504C       59                FCC     'YAMAUCHI'
10710   5054       93                FCB     $93
10720   5055       D38F              FDB     $D38F
10730   5057       90                FCB     $90
10740                         *
10750   5058 AE    8D 0008           LDX     SADDR,PCR
10760   505C EE    8D 0006           LDU     EADDR,PCR
10770   5060 BD    E2F4              JSR     $E2F4
```

```
        5063 39                     RTS
                               *
        5064          0002     SADDR  RMB    2
        5066          0002     EADDR  RMB    2
                               *
                      5068     ESERS  EQU    *
                               *
        5068          0002     STLINE RMB    2
        506A          0002     LINES  RMB    2
                               *
                      5000            END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=506B
PROGRAM ENTRY ADDR=5000
```

図3・3・11　ＥＲＳＶＲＭのフローチャート



ＥＲＳＳＯＦ

ＥＲＳＳＯＦ　　$Ｅ３２５　～　$Ｅ３４７
ＥＲＳＳＵＢ　　$Ｅ３２８

機　能　画面の特定の行を消去します（ＶＲＡＭおよびコンソールバッファの両方を消去します）．ＥＲＳＳＯＦでは，消去の間は画面を表示しませんが，ＥＲＳＳＵＢでは，消去している間でも画面を表示したままにすることができます．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

入力情報　．ＣＶＴＯＰ　第１バイト目（$Ｄ０７Ｄ）に消去開始行
第２バイト目（$Ｄ０７Ｅ）に消去終了行の次の行

ワークエリア　ＣＳＬＣＤＳ　$Ｄ０２３　単色表示フラグ
．ＣＶＴＯＰ　$Ｄ０７Ｄ～$Ｄ０７Ｅ　消去行情報

サブルーチン　ＥＲＳＶＲＭ　$Ｅ２Ｄ３　特定の行を消去（ＶＲＡＭ）
ＥＲＳＢＵＦ　$Ｅ３４１　特定の行を消去（コンソールバッファ）

解　説　動作は次のとおりです．

1）ＣＲＴ表示フラグ（$Ｄ４０８）をＯＦＦにします（ＥＲＳＳＵＢではしません）．
2）単色表示のときに，他の画面も同時に消去するように，ＣＳＬＣＤＳを符号反転してＥＲＳＶＲＭを呼び出します．呼び終った後は，もう一度符号反転し，元の値に戻します．
3）ＥＲＳＢＵＦに分岐して，コンソールバッファを消去します．

次の例は１０～１２行の間を消去します(コンソールバッファおよびＶＲＡＭ)．

```
LDA  #10
LDB  #13
STD .CVTOP
JSR  ERSSOF
```

CSLCDSをNEG（符号反転）をとることについて

CSLCDS（単色表示フラグ）は，単色表示モードでは＄FFの値をとるので，そのままの形でERSVRMを使用すると，緑の画面しか消去されません．単色表示モードでも，他の画面（青と赤）についても消去させるようにするため，CSLCDSのNEGをとって，＄01にしています．ERSVRMではCSLCDSのビット7を見て他の画面を消去するかどうかを判断しています．また，カラーモードのときは，CSLCDSは0の値をとり，0のNEGは0なので問題はありません．なお，スクロールではこの動作をしていないため，単色表示モードでも青と赤の画面は消去されません．単色表示モードは，テキスト表示を速くするのが目的ですから，あえて他の画面を消すのを止めて，スピードをあげているのです．従って，単色表示モード時，スクロール後に青および赤の画面が見づらくなるのは，テキスト表示の高速化という目的に照らし合わせれば，やむをえないことなのです．下記のリストで，CONSOLE文のコンソールカラースイッチを0とした場合と，1とした場合で比較してみてください．カラーモードのときは，他の画面も消去しています．

```
10 WIDTH 80,25
20 CONSOLE 0,25,,1
30 LINE(0,0)-(640,200),PSET,2
40 FOR I=0 TO 100
50   PRINT :NEXT
60 GOTO 30
```

ERSBUF

＄E331 ～ ＄E347

機　能　キャラクタバッファおよびアトリビュートバッファの特定の行を消去します．ただし，VRAM上にあるキャラクタのフォントは消去しません．

レジスタ　A，B，X，Y，U

入力情報　.CVTOP　ERSSOFルーチンと同じ内容

ワークエリア　..ACL　＄D02E　現在のアトリビューコード
.CVTOP　＄D07D～＄D07E　消去行情報

サブルーチン　BADRL　＄E208　コンソールバッファのアドレスを求める
ERSARA　＄E348　特定の領域を同一データで埋める

解　説　動作内容は次のようになります．

1） ＢＡＤＲＬルーチンを用いて，消去開始行および消去終了行の次の行に対するキャラクタバッファとアトリビュートバッファ内のアドレスを各々求めます．

2） 消去開始行および終了行の次の行のキャラクタバッファアドレスから，それらの示すアドレス間を，ＮＵＬＬ（＄００）で満たします（キャラクタバッファの消去）．

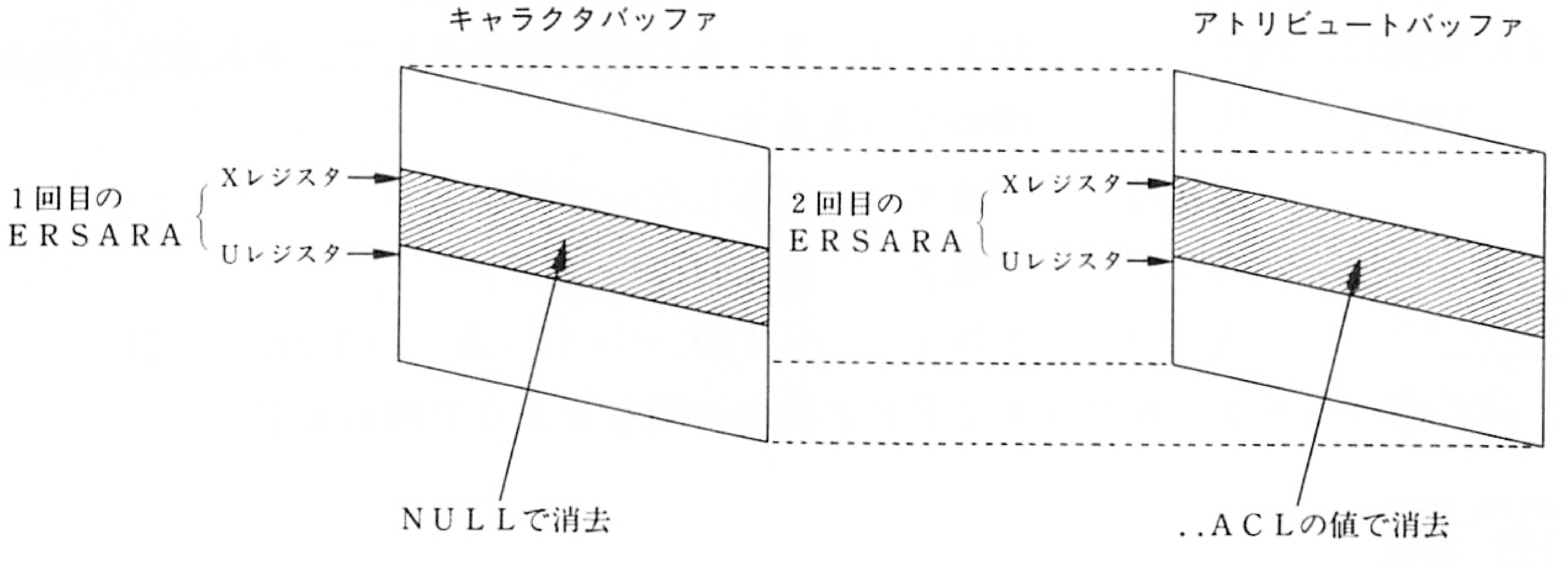
3） 消去開始行および終了行の次の行のアトリビュートバッファアドレスから，それらの示すアドレス間を，ワークエリア..ＡＣＬ（＄Ｄ０２Ｅ）の内容で満たします（アトリビュートバッファの初期化）．

2）～3）では，サブルーチンＥＲＳＡＲＡを用いています．ＥＲＳＡＲＡに渡すＸレジスタ，Ｕレジスタは図３・３・12のようになります．

　次の例は，１０～１２行の間のコンソールバッファをクリアします．

```
LDA  #10
LDB  #13
STD  .CVTOP
JSR  ERSBUF
```

図３・３・12　バッファの消去



ＥＲＳＡＲＡ

＄Ｅ３４８　～　＄Ｅ３６７

機　　能　Ｘレジスタの示す番地から，Ｕレジスタの示す番地の前までの領域を，Ａレジスタの内容で埋めます．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

入力情報　Ａレジスタ　　転送する内容
　　　　　Ｘレジスタ　　先頭番地
　　　　　Ｕレジスタ　　終了番地の次の番地

解　　説　使用するにあたって，次のような制約が課されます．

1）Ｘレジスタは Ｕレジスタより小さいこと　　　　X＜U
2）転送される（Ａレジスタの内容で埋められる）　　(U－X)mod 2＝0
　領域のバイト数は，偶数でかつ6以上であること　　U－X≧6

動作は，次のようになります．

1）Ａレジスタの内容をＢ，Ｘ，Ｙレジスタに複写して，ＰＳＨＵという形で最終番地から転送していきます．

```
        A→B ： D→X：    D→Y        (複写)
   ┌→ PSHU  D, X, Y              (6バイトずつ転送)
   └─ if  U≧(先頭番地＋6)  then  繰り返す
```

2）最後は，残りのバイト数が，０，２，４の場合分けをして，それぞれの場合に応じて，次のように埋めていきます．

残りのバイト数0　　何もしないで終了
残りのバイト数2　　ＳＴＤ，－－Ｕ
残りのバイト数4　　ＳＴＤ，－－Ｕ　＆　ＳＴＤ，－－Ｕ

次の例は，＄００００～＄２ＦＦＦ番地の間を＄８０で埋めます．

```
BITA VACC
LDA  #$80
LDX  #$0000
LDU  #$3000
JSR  ERSARA
STA  VACC
```

ＥＲＳＶ

＄Ｅ３６８　～　＄Ｅ３８２

機　能　ＶＲＡＭ上の，Ｘレジスタの示す番地からＵレジスタの示す番地の手前までの領域を＄００また＄ＦＦで埋めていきます．＄００の場合は，その領域のドットがすべて消えます．また，＄ＦＦの場合はその領域のドットがすべてつきます．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	Ａレジスタ	ビット０＝０のときは＄００，ビット０＝１のときは＄ＦＦで埋めていく	
	Ｘレジスタ	開始番地	（＄００００～＄３ＦＦＦ）
	Ｕレジスタ	終了番地の次の番地	（＄０００１～＄４０００）
出力情報	Ｘレジスタ	開始番地＋＄４０００	
	Ｕレジスタ	終了番地の次の番地＋＄４０００	
サブルーチン	ＥＲＳＡＲＡ	＄Ｅ３４８	特定の領域を同一データで埋める

解　説　このルーチンは，サブルーチンとしてＥＲＳＡＲＡを用いているため，ＥＲＳＡＲＡと同じ制約が課されます．

動作は，次のようになります．

１）Ａレジスタを右にシフトして，ビット０の内容をＣフラグに移します．

２）Ａ，Ｘ，Ｕをスタックに退避した後，Ａレジスタに次のように代入します．

　　Ｃフラグ＝１　　Ａ←＄ＦＦ

　　Ｃフラグ＝０　　Ａ←＄００

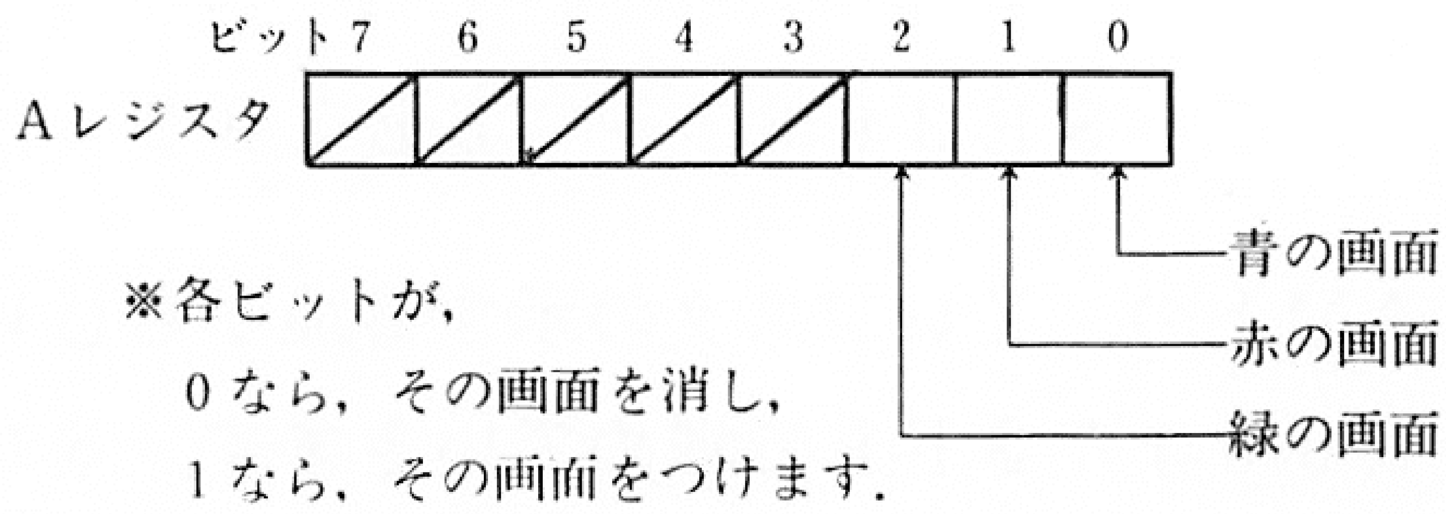
３）ＥＲＳＡＲＡを呼び出して，消去します．この間，ＶＲＡＭアクセスフラグ（＄Ｄ４０９）をオンにしています．

４）スタックから，Ａ，Ｘ，Ｕをとり出し，ＸとＵに＄４０００を加えます．

このルーチンを続けて呼んだ際に，自動的に次のＶＲＡＭがアクセスされるように，ＸレジスタとＵレジスタを更新しています．よって，このルーチンを３回続けて呼ぶ場合，Ａレジスタに図３・３・13の内容を入れておけば，それぞれのＶＲＡＭについて，ドットを消すかつけるかが選択でき，背景の色を指定することができます．

下のサンプルは，３回続けてこのルーチンを呼ぶものですが，呼ぶ際にＸ，Ｕレジスタを少し変えています．

図3・3・13　3回続けて呼ぶときの情報




```
                         *
                         ******************************************
                         *
                         *   graphic test using ERSV
                         *
                         *
                                NAM     S.ERSV
                                TTL     SAMPLE FOR ERSV
                                OPT     MEM,NOGEN,NOW
     5000                       ORG     $5000
                         *
               E368      ERSV   EQU     $E368
                         *
               5000      START  EQU     *
     5000 30   8D 001A          LEAX    S.ERSV,PCR
     5004 C6   30               LDB     #END-S.ERSV
     5006 CE   FC80             LDU     #$FC80
               5009      LOOP1  EQU     *
     5009 B6   FD05             LDA     $FD05    * halt subsystem
     500C 2B   FB   5009        BMI     LOOP1
     500E 86   80               LDA     #$80
     5010 B7   FD05             STA     $FD05
               5013      LOOP2  EQU     *
     5013 A6   80               LDA     ,X+
     5015 A7   C0               STA     ,U+
     5017 5A                    DECB
     5018 26   F9   5013        BNE     LOOP2
     501A 7F   FD05             CLR     $FD05
     501D 39                    RTS
                         *
                         *  program
                         *
               501E      S.ERSV EQU     *
     501E      0000             FDB     $0000
     5020      3F               FCB     $3F
     5021      59               FCC     'YAMAUCHI'
     5029      93               FCB     $93
     502A      D38F             FDB     $D38F
     502C      90               FCB     $90
                         *
     502D 86   07               LDA     #7
     502F 8E   0000             LDX     #$0000
     5032 CE   1900             LDU     #$1900
     5035 BD   E368             JSR     ERSV
                         *
     5038 30   89 0C80          LEAX    $0C80,X
     503C 33   C9 1900          LEAU    $1900,U
     5040 BD   E368             JSR     ERSV
                         *
     5043 30   89 1900          LEAX    $1900,X
     5047 33   C9 0C80          LEAU    $0C80,U
     504B 7E   E368             JMP     ERSV
                         *
               504E      END    EQU     *
                         *
               5000             END     $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000
PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=504D
PROGRAM ENTRY ADDR=5000
```

3－4　PUT，GET

3－4－1　PUT　　　$03　　（$E5E1）

文字列を画面に表示します．また，画面を制御するオーダコードを実行する場合もあります．

共有RAM上のパラメータ

アドレス	記　号	内　　容	値
D383	N	データバイト数	今回転送する文字列のバイト数(N≦124)
D384 ～ D383＋N	String	出力文字列	今回出力する文字列（オーダを含む）

主なサブルーチン

$E51B	CDSPB	キャラクタのフォントの表示（CDSP）
$E678	PUTCHR	一文字出力
$E6E4	PUTATR	アトリビュートの設定
$E745	PUTDAT	データの読込み
$E7BA	PUTINC	文字を出力する位置の更新
$E7DE	PUTDW	文字を出力する垂直位置の更新
$E82E	SCROLL	スクロール（アップ）
$E7FB	ESCSEQ	エスケープシーケンスの処理

補助ワークエリア

$D0A0	PCRORD	CRオーダが実行された（≠0）ことを示すフラグ
$D0A1	PUTGET	PUT（＝0）かGET（≠0）かを示すフラグ
$D0A2 ～$D0A3	PUTIX	共有RAM上の出力文字列を読み込むためのポインタ
$D0A4	PUTFLD	未使用（機能がPUTCFGに吸収されたので不要になりました）

解　説　ＧＥＴの文字列を出力する部分も，ＰＵＴのルーチン（＝＄Ｅ５Ｅ４：ＰＵＴ０００）で行っています．ただし，ＧＥＴの方は，文字列の出力が終ると＄Ｅ９３０（ＧＥＴ０００）にジャンプし，その処理系で画面（キーボード）からの入力を行っています．

ＰＵＴは，次のような操作の繰返しによって，共有ＲＡＭ上にある出力文字列を表示していきます．この繰返しは，転送する文字列がなくなるまで，行われます．

1）共有ＲＡＭ上にある出力文字列から，今回出力する文字のコードを持ってきます(ＰＵＴＤＡＴ)．

2）文字コードがキャラクタ（＄２０～＄７Ｅ，＄８０～＄ＦＦ）なら，その文字を表示します(ＰＵＴＣＨＲ)．表示後は，次に表示する位置を設定します（ＰＵＴＩＮＣ)．

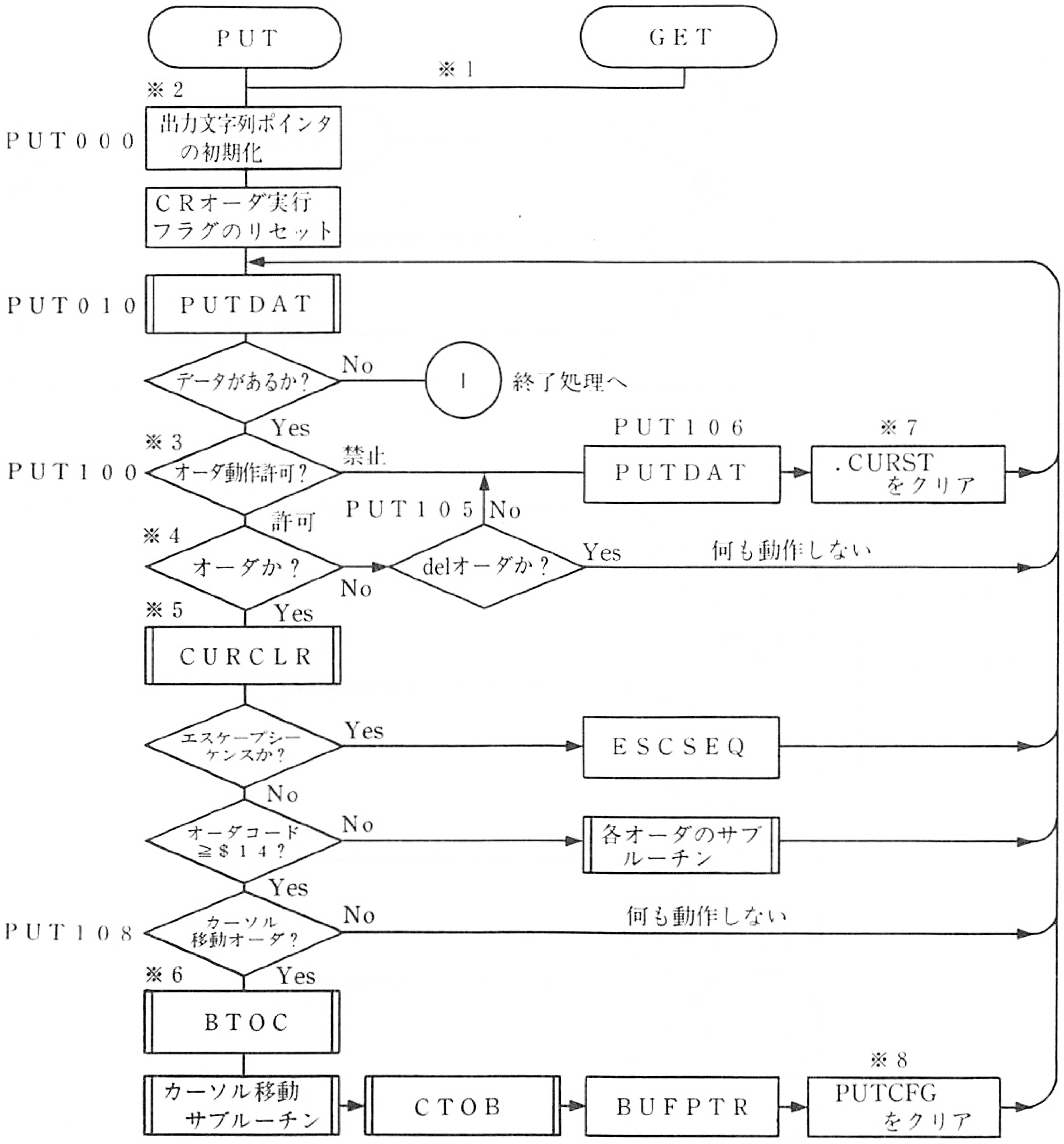
3）文字コードがオーダ（＄００～＄１Ｆ，＄７Ｆ）で，かつオーダ動作が許可されていれば，そのオーダを実行します．オーダの実行は，ジャンプベクトルテーブルに従って各オーダのルーチンを呼ぶことで，行っています．ただし，エスケープシーケンス（＄１Ｂに続くオーダシーケンス）は，ＥＳＣＳＥＱに分岐して行っています．また，カーソルの移動についても，一般のオーダとは別に行っています．

ＰＵＴの動作の概要は，上記の動作の繰返しなのですが，その他の詳細な動作は，図３・４・１～２のフローチャートを参考にして下さい．

各オーダのサブルーチンのアドレスは，図３・４・３のとおりです．これらのサブルーチンへのジャンプベクトルは，＄Ｅ９０８（ＰＵＴＯＲＤ）～＄Ｅ９２Ｆの間に書かれています．ただし，エスケープ（＄１Ｂ）に関しては，ＥＳＣＳＥＱで行っていますし，カーソルの移動を行うものは，ＧＥＴのジャンプベクトル（ＧＥＴＪＣＳ）を用いています（よって，ＧＥＴの処理系のサブルーチンを使用していることになります）．

また，表示後はＥＳＣキーによるプットウェイトの処理を行っています．

図3・4・1　PUTの処理　①



※1　PUTかGETかを示すために，PUT000に入る前にPUTGETフラグ（$D0A1）を設定しています（PUTのとき＝0／GETのとき≠0）.

※2　ポインタの初期化は，PUTIXに#$D384を代入しています.

※3　オーダ動作の許可状態を調べるのは，.CSCTL（$D021）のビット1のオーダ動作フラグを見ています.

※4　オーダかどうかは，文字コードが$20より小さいか否かによって判断しているため，$7Fであるdelについては後で検討しなければなりません.

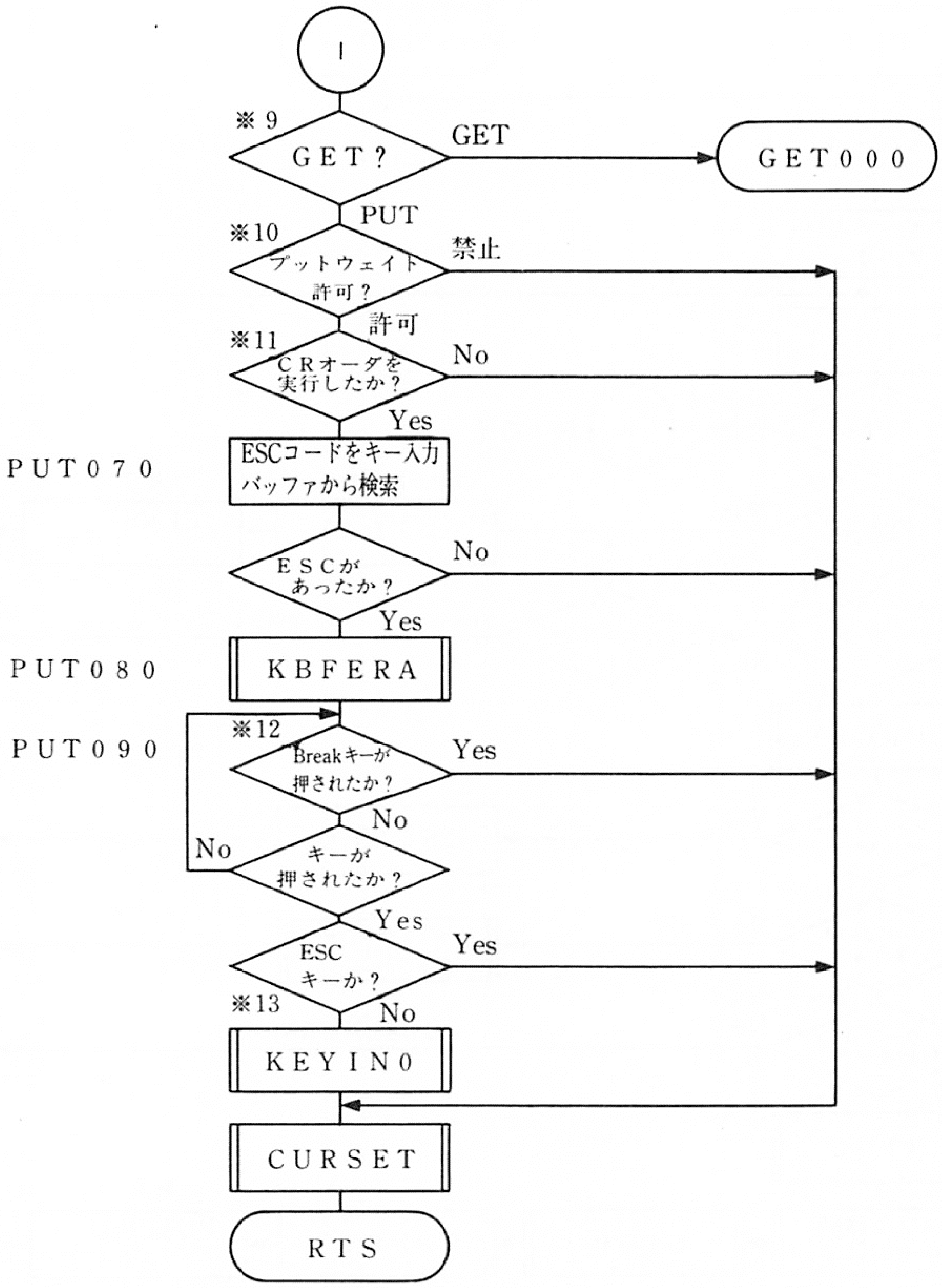
※5　NMI処理ルーチンにおいて，カーソルのある位置へキャラクタのフォンドまたはカーソルを出力しているため，それを禁止させています.

※6　カーソル移動サブルーチンは，カーソルのXY座標を移動するため，バッファポインタの内容をカーソルのワークエリアに代入してから，サブルーチンを呼び出し，変更されたワークエリアの内容を再びバッファポインタに戻しているのです.

※7　カーソルによって，リバース表示させないようにしています.

※8　バッファポインタがフィールドの先頭にきたときに，フィールドの先頭を示すフラグをクリアしないようにしています.

図3・4・2　PUTの処理　②



※9　PUTGETフラグ（$D0A1）を見て判断しています．

※10　CSCTL（$D021）のビット4のプットウェイトフラグで判断しています．

※11　PCRORDフラグ（$D0A0）で判断しています．PCRORDフラグは，CRオーダが実行されたときにPUTOCRサブルーチンによってセットされます．

※12　..ABRT（$D000）によって判断しています．ワークエリア..ABRTは，Breakキーが押されるとIRQ割込み処理ルーチン（IRQAB：$E06E）によってセットされます．

※13　ESCキーでないとき，KEYIN0を行うのは，キー入力バッファのポインタを更新するためです．ESCキーのときは，ポインタの更新を行いませんから，次回PUTをリクエストしたときに，PUT070においてESCコードが検出されることになります．

図3・4・3　PUTのオーダのサブルーチン

オーダコード	アドレス	ラベル名	オーダコード	アドレス	ラベル名
$00～$04	$E6AB	PUTNOP	$0E～$10	$E6AB	PUTNOP
$05	$E6A0	PUTOEL	$11	$E6CE	PUTOSF
$06	$E6AB	PUTNOP	$12	$E700	PUTOSB
$07	$E6A8	PUTOBL	$13	$E731	PUTORC
$08	$E76B	PUTOBS	$14～$1A	――	何も動作しません
$09	$E793	PUTOTB	$1B	$E7FB	ESCSEQ
$0A	$E6BE	PUTOLF	$1C	$EA14	CSRL
$0B	$E6C6	PUTOHM	$1D	$EA1F	CSLL
$0C	$E6AC	PUTOCL	$1E	$EA29	CSUL
$0D	$E6B4	PUTOCR	$1F	$EA37	CSDL

※PUTNOPは，何もしないでただリターン（RTS）するだけです．
$14～$1Aおよび$7Fについては，ジャンプベクトルが設けられていませんし，
何も動作しません．

PUTCHR

PUTSPC　　$E676　～　$E69F
PUTCHR　　$E678

【機　能】 バッファポインタの示す位置に，文字（一文字）を出力します（バッファおよび画面の両方へ）．PUTSPCでは，出力する文字は空白です．

【レジスタ】 A，B，X，Y，U　（A，B，Xは保存）

【入力情報】 Aレジスタ　　出力する文字のコード

【ワークエリア】 PUTCFG　$D022　PUT継続フラグ
..ACL　　$D02E　現在のアトリビュートコード
.BPAD　　$D047～$D048　バッファアドレス
.BAAD　　$D049～$D04A　バッファアドレス

【サブルーチン】 CDSPB　$E51B　キャラクタフォントの出力
PUTATR　$E6E4　アトリビュートの設定
PUTINC　$E7BA　バッファポインタの更新

【解　説】 PUT処理系においては，バッファポインタに基づいて，文字を出力していきます．バッファポインタのXY座標やアドレスは，$D045～$D

０４Ｃ番地の領域に設定されていますので，実際にはこれらのワークエリアを参照して各文字を出力しています．これが，ＰＵＴ処理系でのバッファポインタの役割になっています．カーソルのＸＹ座標を基準として各処理を行っているＧＥＴ処理系とは対照的に，ＰＵＴ処理系ではバッファポインタを中心に各処理を行っています．

動作は次のようになります．

1)　．ＢＰＡＤ（バッファポインタのキャラクタバッファ内のアドレス）の示すアドレスへ，Ａレジスタの内容を格納することによって，キャラクタバッファへの出力を行います．

2)　アトリビュートバッファへの出力は，次のように行います．

　i）出力する座標（．ＢＡＡＤがそのアドレスを示す）がフィールドの先頭でないときは，現在のアトリビュートコード（..ＡＣＬ）を．ＢＡＡＤの示すアドレスへ出力します．

　ii）フィールドの先頭であったとき，ＰＵＴ継続フラグ（ＰＵＴＣＦＧ）がセット（＝＄ＦＦ）されていれば，ＰＵＴＡＴＲルーチンを用いて，そのフィールドを現在のアトリビュートコードで満たしてから，．ＢＡＡＤの示すアドレスへ..ＡＣＬの値を出力します．

　iii）フィールドの先頭で，ＰＵＴ継続フラグがクリアされていた場合は，現在のアトリビュートコードのＦ（フィールドの先頭）フラグ（＝ビット７）をセットしてから，．ＢＡＡＤの示すアドレスへ出力します．

3)　ＣＤＳＰＢを用いて，キャラクタのフォントを出力します．

4)　ＰＵＴＩＮＣで，バッファポインタの位置を更新させます．また，ＰＵＴ継続フラグをセットしておきます．

ワークエリアＰＵＴＣＦＧについて

ＰＵＴ継続フラグＰＵＴＣＦＧ（＄Ｄ０２２）は，前述したようにフィールドの先頭にバッファポインタがきた場合，そのフィールドを矯正するかしないかということを示しています．

ＰＵＴ継続フラグがセットされている（＝＄ＦＦ）ときは，矯正を行います．具体的には，フィールドの先頭を示すＦフラグをクリアして，現在のアトリビュートコードを設定します．そして，そのフィールドの最後までを現在のアトリビュートコードで埋めてしまいます．

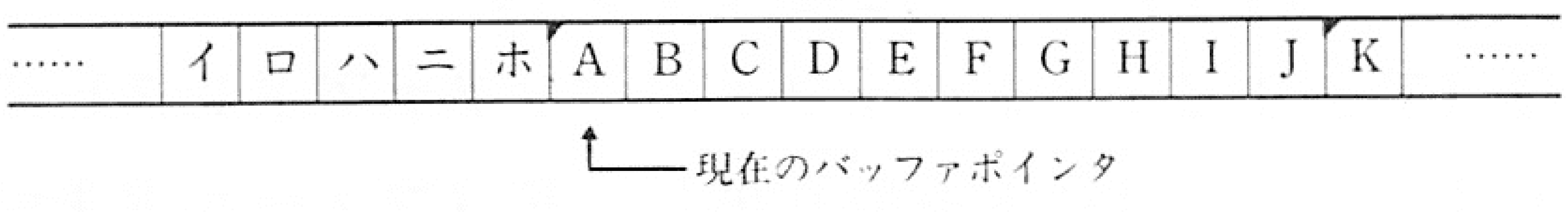
ＰＵＴ継続フラグがクリアされている（＝＄００）ときは，矯正を行いませ

ん．Ｆフラグがセットされたまま，現在のアトリビュートコードが設定されます．そして，そのフィールドのアトリビュートコードは，文字を出力した所までは，現在のアトリビュートコードになりますが，それ以降の場所は元のアトリビュートのままになっています．
一般に文字を出力していくときは，ＰＵＴ継続フラグはセットされているのですが，カーソル移動，画面消去，改行，１行削除などのオーダが実行された後は，Ｆフラグをクリアするのを防止するために，クリアされています．
また，ＧＥＴ処理系においても，同様の理由からクリアされています．

図3・4・4　ＰＵＴＣＦＧによる処理の相違

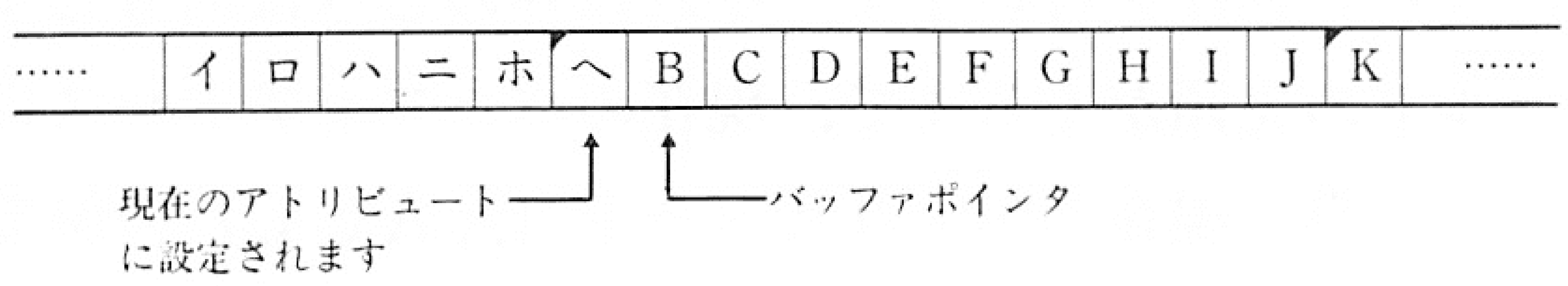
○例えばバッファの内容が下のようになっていて，バッファポインタがフィールドの先頭を指しているとします（◤はフィールドの先頭を示します）．

バッファの内容



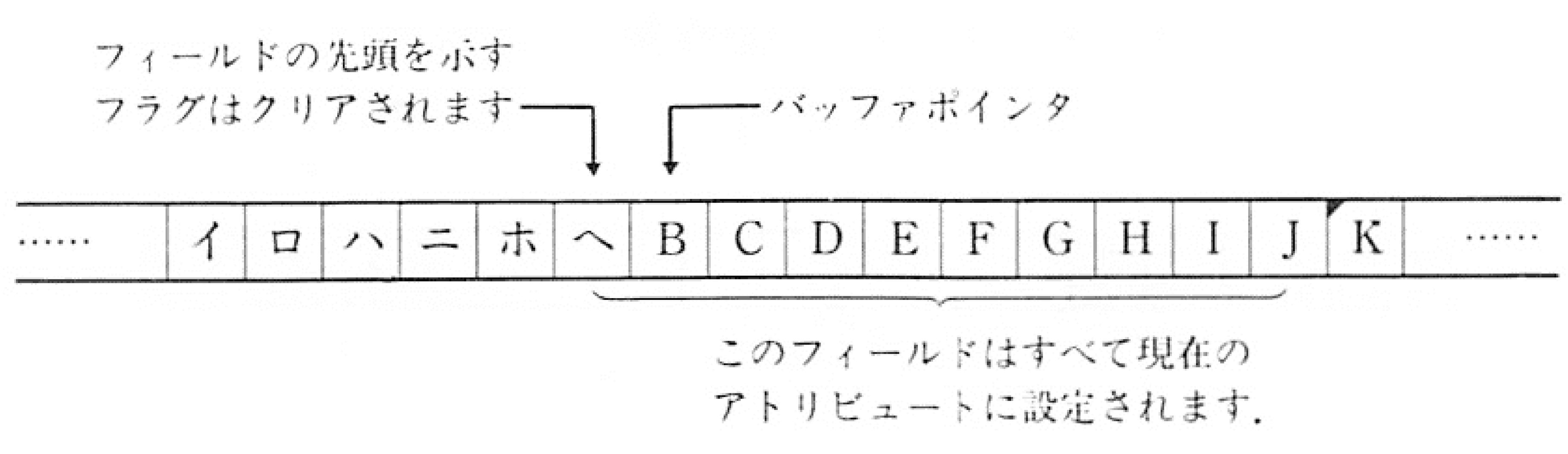
○次に，ＰＵＴＣＨＲにおいて，１文字"へ"を出力するときは次のようになります．

● ＰＵＴ継続フラグ（ＰＵＴＣＦＧ）がクリアされているとき．



※フィールドの先頭を示すフラグ（Ｆフラグ）は残ります．

● ＰＵＴ継続フラグ（ＰＵＴＣＦＧ）がセットされているとき．



ＰＵＴＡＴＲ

＄Ｅ６Ｅ４　～　＄Ｅ６ＦＦ

機　能　バッファポインタの示す位置のアトリビュートと現在のアトリビュートが異るとき，次のフィールドの先頭まで，現在のアトリビュートで満たしていきます．

レジスタ　Ａ，Ｂ，Ｘ　（Ａ，Ｂは保存）

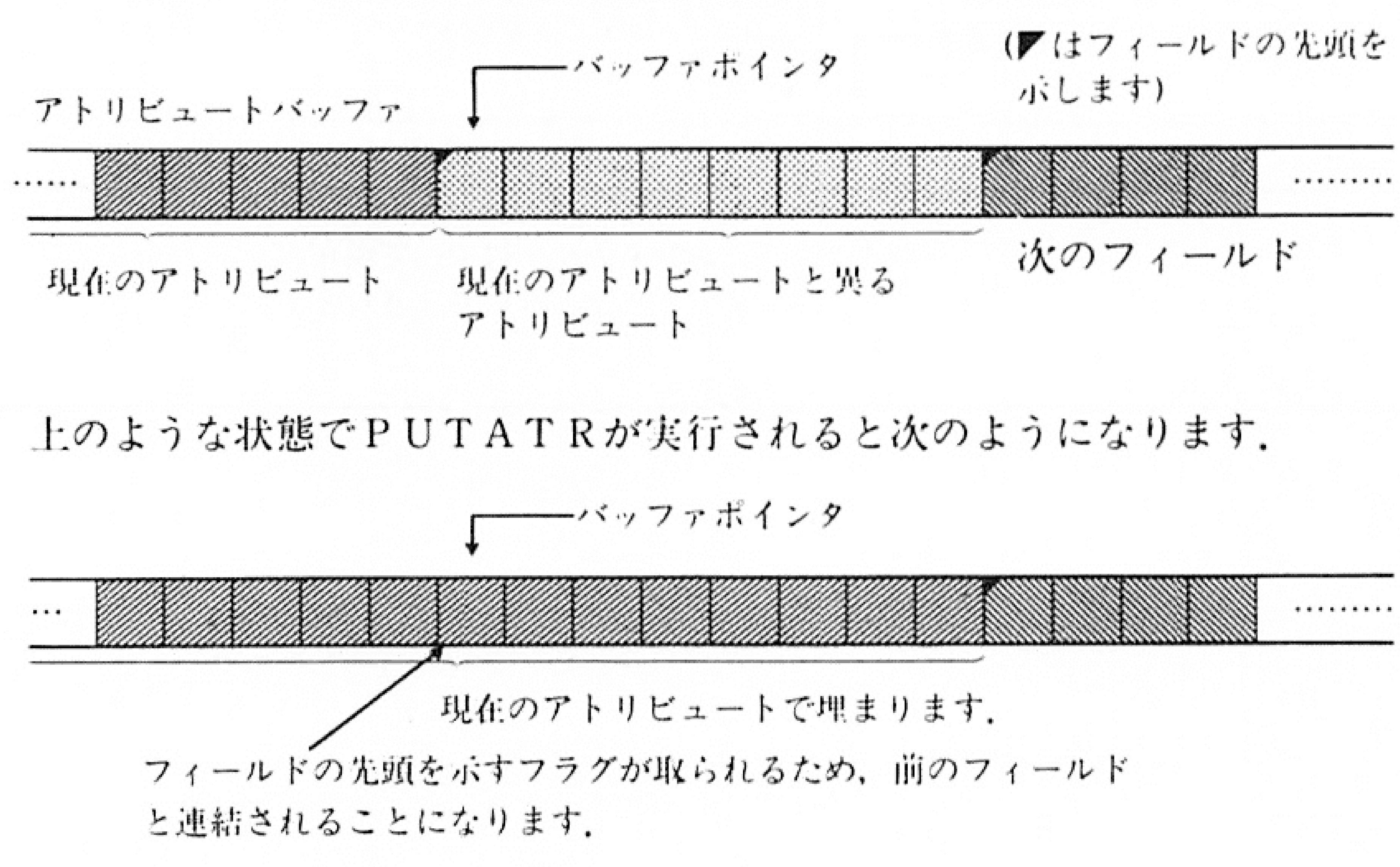
ワークエリア　ＣＳＬＣＤＳ　＄Ｄ０２３　単色表示フラグ
..ＡＣＬ　＄Ｄ０２Ｅ　現在のアトリビュートコード(カラー)
.ＢＡＡＤ　＄Ｄ０４９～＄Ｄ０４Ａ　バッファアドレス

解　説　次のように動作します．バッファアドレスというのは，バッファポインタが示すバッファ内のアドレスを意味します．

1)　.ＢＡＡＤ（アトリビュートバッファのバッファアドレス）の示すアドレスから，アトリビュートコードを取り出し，Ｆ（フィールド先頭）フラグとＭ（変更）フラグをクリアして元のアドレスに戻します．
2)　そのアトリビュートコードと..ＡＣＬ（現在のアトリビュートコード）の内容を比較し，その値が異っていて，かつカラーモード（ＣＳＬＣＤＳ＝０）のときは，３）の動作を行います．それ以外のときは何もしません．
3)　次のフィールドの先頭がくるまで，現在のバッファアドレス以降のアトリビュートバッファを，現在のアトリビュート（..ＡＣＬ）の値で埋めていきます．

ＰＵＴＡＴＲは，このように，分割されているフィールドを１つのフィールドにまとめるときに用いられています．

図３・４・５　ＰＵＴＡＴＲの動作例

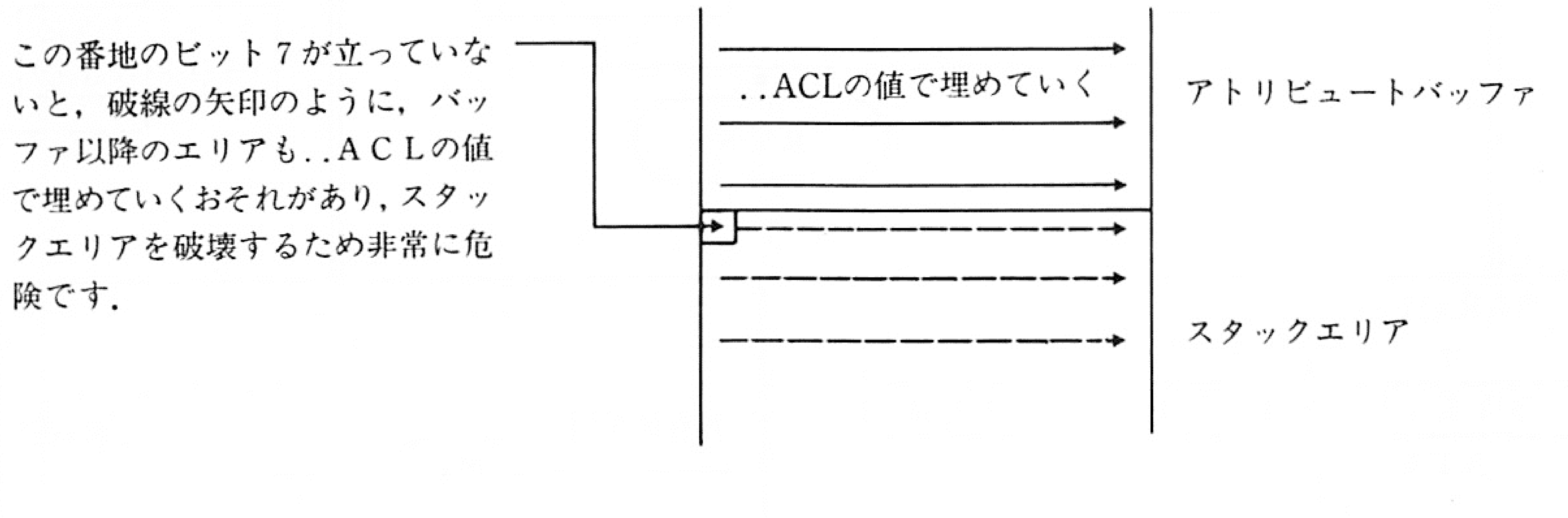


ワークエリア..ＡＣＬ（＄Ｄ０２Ｅ）について

..ＡＣＬには，現在のアトリビュートコードが入っています．ただし，アトリビュートコードの下位６ビットだけが格納されていて，Ｆ（フィールドの先頭）フラグ（＝ビット７）とＭ（変更）フラグ（＝ビット６）はいつもクリアされています．ワークエリア..ＡＣＬは，ＰＵＴのＳＦオーダやＥＲＡＳＥ２コマンドによって設定され，１文字出力する際などにアトリビュートバッファに格納されます．また，画面の消去を行えば，すべてのアトリビュートバッファが..ＡＣＬの内容で満たされますし，ＰＵＴＡＴＲルーチンでは，一つのフィールドを..ＡＣＬの内容で埋めていきます．

ＥＲＳＡＬＬ，ＡＳＭＦＬＤ（３－３－２節）において，アトリビュートバッファの終了アドレスの次の番地のビット７を立てているのは，ＰＵＴＡＴＲにおいて，..ＡＣＬの内容でアトリビュートバッファを埋めていくループを終了させるために行っているのです．その番地のビット７が立っていないと，アトリビュートバッファ以降の場所にも..ＡＣＬの値で埋めていってしまうからです．アトリビュートバッファの後は，スタックエリアが控えており，スタックエリアを破壊してしまいます．そのために，この番地のビット７は重要な役目を果たしています．

図３・４・６　アトリビュートバッファの次の番地のビット７

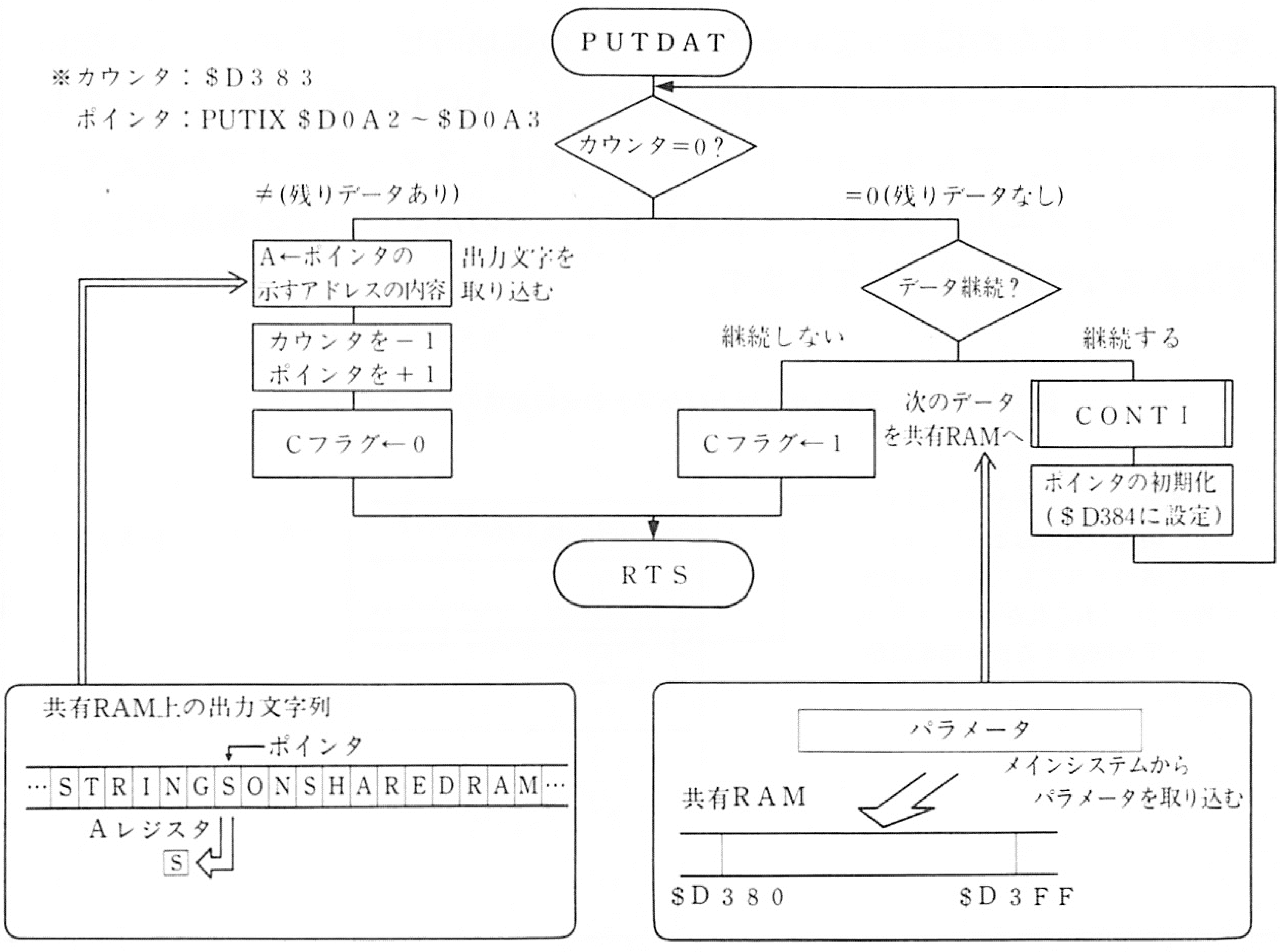


ＰＵＴＤＡＴ

＄Ｅ７４５　～　＄Ｅ７６Ａ

機　能	ＰＵＴ（ＧＥＴ）の共有ＲＡＭ上のパラメータ中の出力文字列(String：＄Ｄ３８４～）中から，１文字をＡレジスタに取り込みます.
レジスタ	Ａ，Ｘ　（Ｘは保存）
出力情報	Ａレジスタ　　出力文字列の一文字
ワークエリア	..ＣＯＮＴ　　＄Ｄ００１　データ継続フラグ
	ＰＵＴＩＸ　　＄Ｄ０Ａ２～＄Ｄ０Ａ３（出力文字列のポインタ）
	共有ＲＡＭ　　＄Ｄ３８３　（Ｎ：データバイト数）
サブルーチン	ＣＯＮＴＩ　　＄Ｅ１２１　継続データをメインから受け取る

図３・４・７　PUTDATの処理



解　説　ＰＵＴコマンドのための共有ＲＡＭ上のパラメータＮ（＄Ｄ３８３：データバイト数）をＰＵＴ処理系では，未出力の文字数を示すカウンタとして用いています．１文字出力していくごとに，Ｎを－１していき，出力文字列のポインタ（ＰＵＴＩＸ）を＋１していきます．
Ａレジスタには，ポインタＰＵＴＩＸの示す，共有ＲＡＭ上の一文字が代入されます．ただし，カウンタが０になっているときは，データが継続しているか（..ＣＯＮＴ≠０かどうか）を調査し，データが継続していれば，ＣＯＮＴＩルーチンを用いて，メインからデータを共有ＲＡＭ上へ読み込ませ，繰り返します．このとき，ＰＵＴＩＸは，＄Ｄ３８４（Stringの先頭）に初期化されます．データが継続していないときは，Ｃフラグをセットして終了します．ＰＵＴのメインルーチンは，Ｃフラグを見て，出力文字列を全部出力し終わったことを確認し，終了します．

ＰＵＴＩＮＣ

＄Ｅ７ＢＡ　～　＄Ｅ７ＤＤ

機　能	バッファポインタを更新します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	.ＣＳＣＴＬ	＄Ｄ０２１	コンソール制御フラグ
	..ＬＮＣ	＄Ｄ０３６	１行の桁数
	..ＶＣＵ	＄Ｄ０３９～＄Ｄ０３Ａ	ＶＲＡＭ上の桁単位
	.ＢＰＸ	＄Ｄ０４６	バッファポインタのＸ座標
	.ＢＰＡＤ	＄Ｄ０４７～＄Ｄ０４８	バッファアドレス
	.ＢＡＡＤ	＄Ｄ０４９～＄Ｄ０４Ａ	バッファアドレス
	.ＢＶＡＤ	＄Ｄ０４Ｂ～＄Ｄ０４Ｃ	ＶＲＡＭ上のアドレス
サブルーチン	ＢＵＦＰＴＲ	＄Ｅ１ＦＣ	バッファポインタのワークエリア設定
	ＰＵＴＤＷ	＄Ｅ７ＤＥ	１行下にバッファポインタを移す

解　説　ＰＵＴ処理系では，バッファポインタの示す位置へ，文字を出力していますから，出力後は，これを更新しなければなりません．更新は，Ｘ座標（キャラクタ単位）を＋１します．これに伴い，バッファアドレスもそれぞれ＋１していきます．また，ＶＲＡＭ上のアドレスも１文字分進めます．具体的には，次のようになります．

1)　ワークエリア .ＢＰＸを＋１します．その結果が１行の桁数（..ＬＮＣ）よ

り大きくなければ，キャラクタバッファのアドレス .BPADとアトリビュートバッファのアドレス .BAADを＋1します．VRAM上のアドレスも，1文字分進めます（VRAM上での桁単位である..VCUを足します）．

2) 1行の桁数と等しいか，それより大きいときは，X座標である .BPXを0にしておいてから，PUTDWに制御を移します．ただし，このときコンソール制御フラグ .CSCTLのビット7が立っていれば，PUTDWに制御が移らず，BUFPTRでバッファアドレスの設定をするだけで終ります．

.CSCTLのビット7をセットしておくと，このようにPUTで出力する文字列が1行以上になってしまうときに，改行が行われず，本来は次の行に表示すべき文字列を，再び同じ行の先頭から表示していくことになります．コンソール制御フラグ .CSCTLは，Console Control コマンドで自由に設定できます．詳しくは，3－4－5節を参照して下さい．

PUTDWに処理が移った後のことは，次のPUTDWを参照して下さい．

PUTDW

$E7DE ~ $E7FA

機　能	バッファポインタのY座標を1つ増します．		
レジスタ	A，B，X，Y，U		
ワークエリア	.CSCTL	$D021	コンソール制御フラグ
	.BPY	$D045	バッファポインタY座標
	.MDRG	$D04D~$D04F	ウィンドウ画面情報
サブルーチン	BUFPTR	$E1FC	バッファポインタのワークエリア設定
	PUTOCL	$E6AC	画面の消去
	SCROLL	$E82E	スクロールアップ
	KEYGET	$FD6E	キー入力待ちとして

解　説　PUTINCがキャラクタX座標を1つ進めるものに対して，このルーチンはバッファポインタのキャラクタY座標を1つ進めるルーチンです．次のような動作をします．

1) Y座標を1つ進めます（ .BPYを＋1します）．その結果，現在選択されているウィンドウ画面の終了行（ .MDRG＋1が保持）より小さい場合は，BUFPTRを用いて，バッファアドレスなどのワークエリアを設定します．

2) 終了行と等しいか，より大きくなった場合は，もし現在のウィンドウ画面が

スクロール画面であれば(.MDRG＋2が1のとき)，SCROLLに制御を移し，スクロールを行ないます．ページ画面であれば，コンソール制御フラグ（.CSCTL）のビット3（ページウェイトフラグ）を見てセットされていればページウェイトをします．ページウェイトの具体的な動作は，BEEP音を出して，KEYGETを用いてキー入力を待ちます．ページウェイトでない場合や，キーが押されたときは，PUTOCLを用いて画面をクリアします．

SCROLL

$E82E ～ $E8F0

機　能	スクロール画面を1行スクロールアップする．		
レジスタ	A，B，X，Y，U		
ワークエリア	.VOFST	$D01F～$D020	オフセット値
	CSLCDS	$D023	単色表示フラグ
	..LNC	$D036	一行の桁数
	..FS	$D038	PFキー表示フラグ
	..VLU	$D03B～$D03C	VRAM上での行単位
	..DMAP+7	$D044	(画面の終了行)
	.BPY	$D045	バッファポインタのY座標
	.MDRG	$D04D～$D04F	ウィンドウ画面情報
	.CVTOP	$D07D～$D07E	消去する行のY座標
サブルーチン	BADRL	$E208	バッファアドレスを求める
	BVADL	$E221	VRAM上のアドレスを求める
	ASMFLD	$E2C5	ウィンドウ画面のフィールド設定
	ERSVRM	$E2D3	特定の行をVRAMから消去
	ERSBUF	$E331	特定の行をバッファから消去
	COPLOP	$E8F1	データのブロック転送
	INDICT	$EE0E	PFキーの表示

解　説　スクロールをするためには，コンソールバッファおよびVRAMをスクロールさせる必要があります．VRAMのスクロールの場合には，スクロール画面が全画面に対してセットされているのであれば，オフセットレジスタによって画面をスクロールさせれば良いのですが，他の画面がある場合は，VRAM

の内容をブロック転送しなければなりません．これらの処理に対して，次のように動作します．

1)　画面のスクロール部

ｉ）全画面がスクロール画面（フルスクロールモード）の場合

画面の終了行の次の行のＶＲＡＭを削除します（これは，ＶＲＡＭの表示されない部分を事前にクリアする目的で行っています）．また，ＰＦキー表示モード（ＦＭ－７においては，フルスクロールモードでＰＦキー表示モードになることはありませんが)のときは，スクロール画面の次の行(＝ＰＦキー表示画面の先頭行）も削除しています．

事前にこれらの行をＶＲＡＭから削除した後は，オフセットレジスタにオフセットを加算します（２章にオフセットとスクロールについての記述があります）．加算する値は，..ＶＣＵ（ＶＲＡＭ上での行単位：３－３－１節参照）の内容を用いますから，１行分上にスクロールアップすることになります．オフセットを加算してスクロールさせた後は，ＰＦキー表示モードのために表示ルーチンＩＮＤＩＣＴを呼び出します．

ii）画面の１部がスクロール画面の場合

スクロール画面の先頭行および終了行（の次の行）の左端のＶＲＡＭ上のアドレスを各々求めます．そして，それらのアドレスで示されるＶＲＡＭの領域の内容を１行分上にブロック転送させます．カラーモードのときは，３画面とも転送を行いますが，単色表示のときは，緑の画面だけ転送を行います．

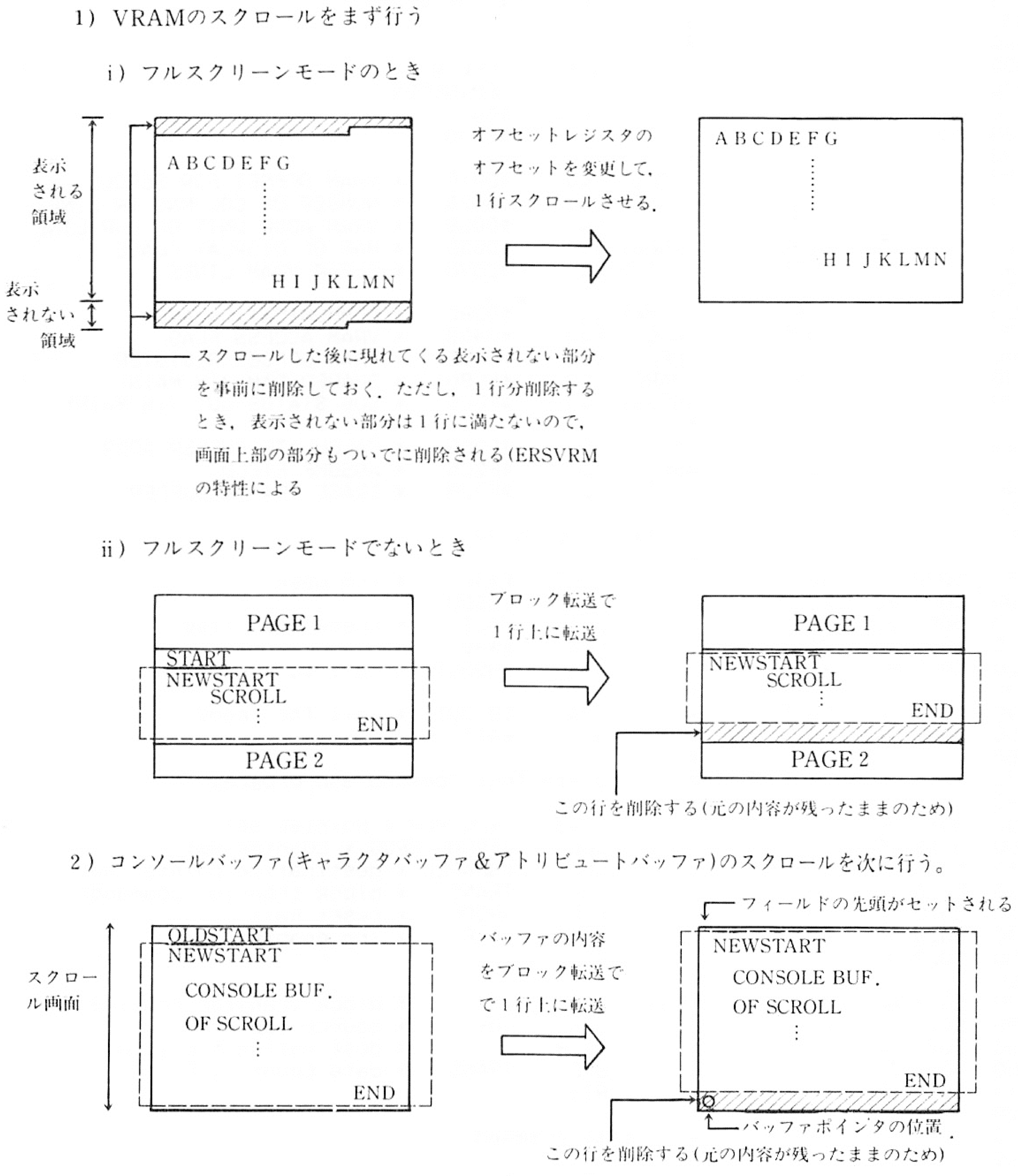
2)　コンソールバッファのスクロール部

スクロール画面の先頭行の次の行から終了行までの領域を１行上にブロック転送します．転送は２回行われ，１回目はコンソールバッファの内容を，２回目はアトリビュートバッファの内容を１行上に転送しています．転送にはＣＯＰＬＯＰを用いています．

3)　終了行の削除

スクロール画面の終了行のＹ座標値を，．ＢＰＹ（バッファポインタＹ座標）に設定しています．フルスクロールモードでないときは，画面の終了行を削除(ＶＲＡＭ，コンソールバッファ共）します．フルスクロールのときは，バッファだけ削除します．最後にＡＳＭＦＬＤを呼び出して，スクロール画面の先頭にフィールドを設定し，バッファポインタからバッファアドレスを求めます．

図3・4・8 SCROLLの処理



※2)の操作は，フルスクリーンモードであるなしにかかわらず行われます．

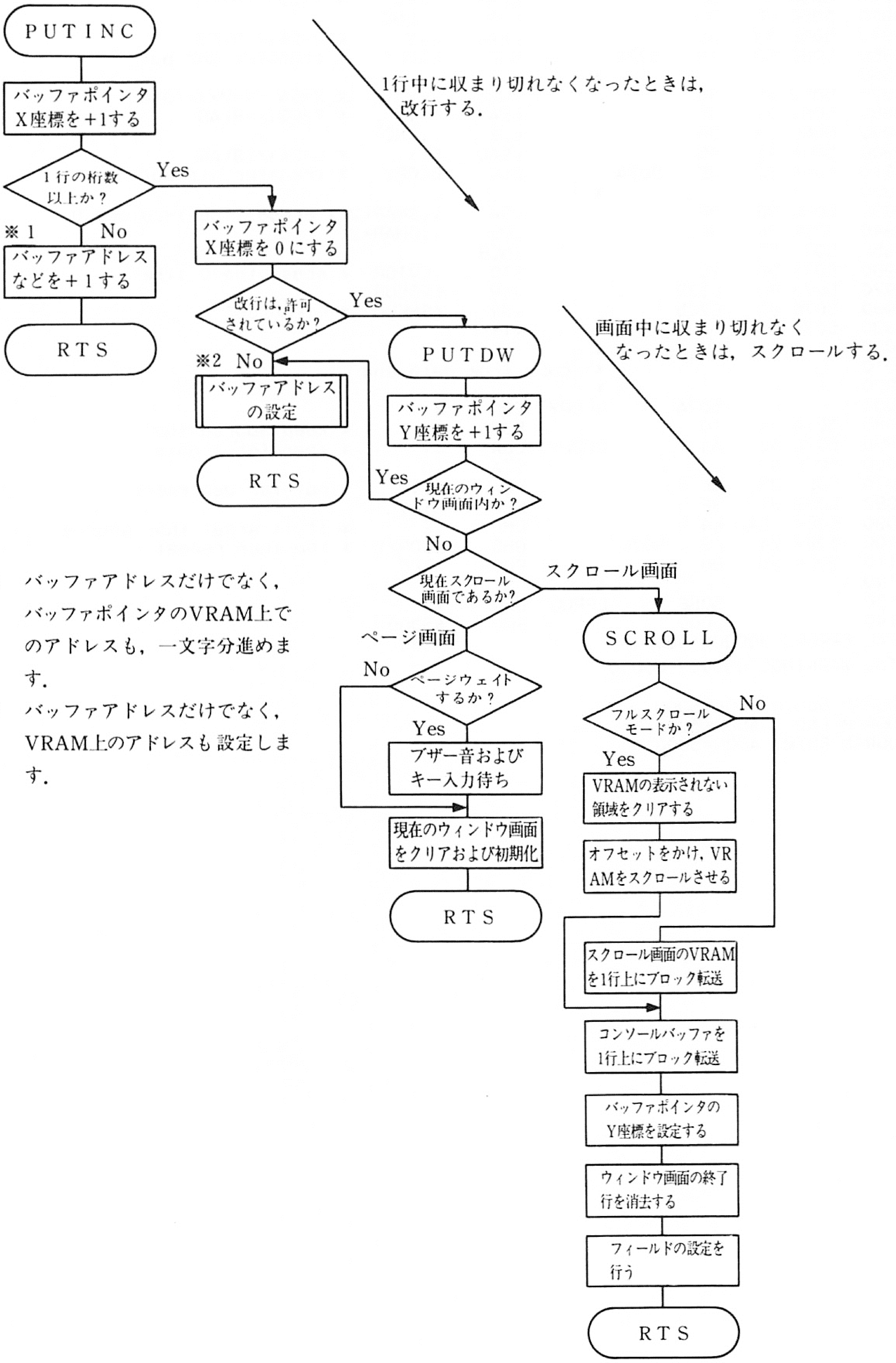
次のサンプルは，画面を1行下にスクロールさせます．ただし，フルスクロールモードでなくても，全画面をスクロールさせています．動作はスクリーンアップ（フルスクロールモード）とだいたい同じですが，スクロール前の表示されない領域の消去は行いません．スクロール後の第0行をバッファ，ＶＲＡＭ共に削除しています（ＥＲＳＳＵＢを用いています）．なお，ＡＳＭＦＬＤを用いていますので，ウィンドウ画面の開始行の左端にフィールドをセットしますから，フルスクリーンモードでないときは注意して下さい．

```
                              **************************************************
                              *
                              *    scroll down program
                              *
                              *
                                   NAM    SCRLDW
                                   OPT    MEM,NOGEN
                   00D0            SETDP  $D0
5000                               ORG    $5000
                              *
                   D01F       .VOFST EQU  $D01F    * VRAM OFFSET FOR SCROLL
                   D036       ..LNC  EQU  $D036    * NUMBER OF COLUMNS IN LINE
                   D03B       ..VLU  EQU  $D03B    * VRAM ADDR UNIT OF CHR LINE
                   D03D       ..DMAP EQU  $D03D    * MAP OF DISPLAY PLACE
                   D07D       .CVTOP EQU  $D07D    * CLEAR VRAM LINES
                              *
                   D380       .COMAR EQU  $D380    * COMMUNICATION AREA
                   D409       VACC   EQU  $D409    * VRAM ACCESS FLAG
                   D40E       SOFFST EQU  $D40E    * VRAM OFFSET REGISTER
                   FC80       SHARED EQU  $FC80    * SHARED RAM (IN MAIN)
                   FD05       SBSOUT EQU  $FD05    * SUB STATUS OUT (IN MAIN)
                              *
                   E208       BADRL  EQU  $E208    * CALCULATE BUFFER ADDR
                   E2C5       ASMFLD EQU  $E2C5    * ASSUME FIELD
                   E328       ERSSUB EQU  $E328    * ERASE VRAM & BUFFER
                              *
                              *  halt subsystem
                              *
5000 1A            40                ORCC   #$40     * irq mask
5002 B6            FD05       HALT   LDA    SBSOUT
5005 2B            FB    5002        BMI    HALT     * check busy flag
5007 86            80                LDA    #$80
5009 B7            FD05              STA    SBSOUT   * halt to subsystem
                              *
500C B6            FD05       WAIT   LDA    SBSOUT   * wait for ready
500F 2A            FB    500C        BPL    WAIT
                              *
                              *  translate Test command and program
                              *
5011 30            8D 0016           LEAX   PROG,PCR * pointer set
5015 C6            60                LDB    #EPROG-PROG * counter set
5017 CE            FC80              LDU    #SHARED  * destination pointer set
501A BD            5023              JSR    TRANS    * block transfer commands
501D 7F            FD05              CLR    SBSOUT   * reset halt
5020 1C            BF                ANDCC  #$BF     * irq enable
5022 39                              RTS
                              *
5023 A6            80         TRANS  LDA    ,X+      * block transfer routine
5025 A7            C0                STA    ,U+      * source top .. X
5027 5A                              DECB            * destination top .. U
5028 26            F9    5023        BNE    TRANS    * data count .. B
502A 39                              RTS
                              *
                              *    test command
                              *
                   502B       PROG   EQU    *
502B               0000              FDB    $0000
502D               3F                FCB    $3F      * test command
502E               59                FCC    #YAMAUCHI# * key word
5036               93                FCB    $93
5037               D38F              FDB    .COMAR+$F
5039               90                FCB    $90
                              *
                              *   scroll down
                              *
503A DC            1F                LDD    .VOFST   * load offset value
503C 93            3B                SUBD   ..VLU    * subtract from offset
503E B5            D409              BITA   VACC     * vram access flag on
5041 DD            1F                STD    .VOFST   * set it to work area
5043 FD            D40E              STD    SOFFST   * set it to i/o
5046 B7            D409              STA    VACC     * vram access flag off
                              *
                              *  block transfer chr & atr buffer
                              *
5049 96            44                LDA    ..DMAP+7 * C means chr buf line addr
504B 4A                              DECA            * A means atr buf line addr
504C BD            E208              JSR    BADRL    * Xreg=24(/19)C Yreg=24(/19)A
504F 34            30                PSHS   X,Y      * keep it
5051 96            3F                LDA    ..DMAP+2
```

```
5053 BD   E208              JSR    BADRL    * X=0C  Y=0A
5056 35   40                PULS   U        * U=24(/19)C
5058 34   20                PSHS   Y        * B means end of line
505A 31   5F                LEAY   -1,U     * Y=23(/18)CB
505C D6   36                LDB    ..LNC
505E 33   A5                LEAU   B,Y      * U=24(/19)CB
5060 8D   18   507A         BSR    DCOPY    * transfer chr buf
                    *
5062 35   30                PULS   X,Y      * X=0A  Y=24(/19)A
5064 31   3F                LEAY   -1,Y     * Y=23(/18)AB
5066 D6   36                LDB    ..LNC
5068 33   A5                LEAU   B,Y      * U=24(/19)AB
506A 8D   0E   507A         BSR    DCOPY    * transfer atr buf
                    *
506C 96   3F                LDA    ..DMAP+2 * erase line Y-co·ord 0
506E D6   3F                LDB    ..DMAP+2
5070 5C                     INCB
5071 DD   7D                STD    .CVTOP   * erase the 0 line
5073 BD   E328              JSR    ERSSUB
5076 BD   E2C5              JSR    ASMFLD   * FIELD SET
5079 39                     RTS
                    *
                    *  copy from data end
                    *
          507A      DCOPY   EQU    *
507A 34   10                PSHS   X        * keep source top
507C A6   A4        DCOPY1  LDA    ,Y       * transfer 1 data
507E A7   C4                STA    ,U
5080 33   5F                LEAU   -1,U     * pointer decrement
5082 31   3F                LEAY   -1,Y
5084 10AC E4                CMPY   ,S       * If it great than source
5087 24   F3   507C         BHS    DCOPY1   * top then repeat
5089 35   90                PULS   X,PC
                    *
          508B      EPROG   EQU    *
          5000              END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=508A
PROGRAM ENTRY ADDR=5000
```

図3・4・9　PUTINC～PUTDW～SCROLLの処理の流れ



※1　バッファアドレスだけでなく，バッファポインタのVRAM上でのアドレスも，一文字分進めます．

※2　バッファアドレスだけでなく，VRAM上のアドレスも設定します．

バッファポインタとカーソルについて

PUT，GET処理系では，画面に文字を出力するポインタとして，バッファポインタとカーソルがあります．バッファポインタはPUT処理系で頻繁に用いられており，文字を出力するためのポインタとしての役割を持っています．カーソルはGET処理で頻繁に用いられており，キー入力した文字を出力するためのポインタとしての役割を持っています．
また，カーソルのPUT処理系での用途は，GET内のサブルーチンを呼びだすときに用いられ，一方，バッファポインタのGET処理系での用途は，画面をどこまで編集したかという，編集を行った最大位置を記憶するためと，一時的にPUTのサブルーチンを呼び出すために用いられています．しかし，これらの役割は補助的なものです．
バッファポインタとカーソルは，その目的の微妙な相違によって，一方の処理系で多く使用されることになります．このように機能の違った2つのポインタを持って画面操作をすることが一般的なようです．
また，バッファポインタとカーソルは，ワークエリアにY座標，X座標，キャラクタバッファアドレス，アトリビュートバッファアドレス，VRAM上のアドレスを各々持っており，それの総称としても用いることがあります．
なおPUT，GETにおいて，バッファポインタがいつも最後の文字を出力した位置を記憶しているのは，次にPUT，GETが用いられたときにその処理の先頭の位置として用いるためです．

バッファアドレスについて

バッファアドレスというのは，特定の座標に対応したコンソールバッファ内のアドレスのことを指します．よって，キャラクタバッファアドレスというのは，キャラクタバッファ内でのアドレスで，アトリビュートバッファアドレスというのは，アトリビュートバッファ内のアドレスを各々意味しています．また，バッファアドレスというのは，バッファポインタ，カーソルに対応してよく用いられているので，「バッファアドレス」という言葉だけで説明しているときは，暗にバッファポインタ，カーソルのバッファアドレスという意味を持つことがありますので注意して下さい．一般には，「バッファポインタのバッファアドレス」という表現を用いています．

終了行，終了アドレスという表現について

サブシステムのＩＮＩＴ，ＥＲＡＳＥ，ＰＵＴ，ＧＥＴの処理系においては，「終了行」，「終了アドレス」という表現が用いられていますが，これらの処理系では，終了行や終了アドレスそのものを扱った処理ではなく，必ず終了行の次の行，終了アドレスの次のアドレスを基準にした処理が行われています．ワークエリアも同様なことが言えます．例えば，ＧＴＢＴＭＣ（＄Ｄ０５２～＄Ｄ０５３）は，ウィンドウ画面の終了キャラクタバッファアドレスですが，実際には終了アドレスそのものではなく，その次の番地のアドレスが入っています．
このように，終了行，終了アドレスと言った場合には，その次の行・アドレスを暗に意味しているので注意して下さい．表現するにあたっては，適宜，「終了アドレスの次のアドレス」という表現を用いています．

ウィンドウ画面について

ウィンドウ画面というのは，２つのページモード画面，スクロールモード画面の３つの画面のうち，現在選択されている画面のことを指しています．現在選択されている状態にあるというのは，バッファポインタやカーソルが，その画面内にあるということになります．例えば，ＧＥＴのスクリーンエディット処理では，カーソルはウィンドウ画面内の範囲でしか移動できません．
ウィンドウ画面を選択するには，バッファポインタをその画面内に移動させればよく，ＰＵＴ処理系のＳＢＡ (Set Buffer Address) オーダによって，バッファポインタを移動したとき，どの画面内にあるかをチェックして，自動的にバッファポインタの含まれている画面が選択されます．
一般的に，ＧＥＴ，ＰＵＴの処理系では，ウィンドウ画面内で処理を行っています．

3－4－2　PUTのオーダサブルーチン

PUTOEL

$E6A0 ～ $E6A7

機　能	現在のバッファポインタの示す位置から，次のフィールドの手前までの文字列を削除します．	
レジスタ	A，B，X，Y，U	
ワークエリア	PUTCFG　$D022	PUT継続フラグ
サブルーチン	BTOC　$E77E	バッファポインタの位置をカーソルに設定
	ELKEY　$EAEC	カーソルがある位置から，次のフィールドの手前までの文字列を削除

解　説　PUT継続フラグをクリアして，現在のバッファポインタがフィールドの先頭にあったときに，次に文字を出力しても，フィールドの先頭フラグがクリアされないようにしています．実際の動作はELKEYにおいて行っていますが，ELKEYはカーソルの位置を基準に処理していますので，ELKEYを呼び出す前に，BTOCを用いてバッファポインタの位置を，カーソルの位置を示すワークエリアにコピーしています．

PUTOBL

GETBEL	$E6A8 ～ $E6AB
PUTOBL	$E6A8
PUTNOP	$E6AB

機　能	ベルを鳴らします．
レジスタ	なし（CCは変わる）

解　説　GETBELは，GET用のエントリで，実際にはPUTOBLと同一のものです．PUTNOPは，PUTのオーダのうち，何も動作を行わないものに対して設けられているエントリで，RTS（$39）によって何もしないで終了します．

ベルの鳴らし方は，I／Oポート$D403(ラベル名BUZZER)をReadすればよいのです．

ＰＵＴＯＣＬ

＄Ｅ６ＡＣ 〜 ＄Ｅ６Ｂ３

機　能	現在のウィンドウ画面を消去します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	ＰＵＴＣＦＧ	＄Ｄ０２２	ＰＵＴ継続フラグ
	.ＭＤＲＧ＋２	＄Ｄ０４Ｆ	現在のウィンドウ画面の画面番号
サブルーチン	ＥＲＡＳＥ１	＄Ｅ２Ａ０	特定の画面の消去

解　説　ワークエリア .ＭＤＲＧ＋２の内容をＡレジスタに代入して，それを入力情報としてＥＲＡＳＥ１を呼び出します．ＥＲＡＳＥ１は，ＥＲＡＳＥ０（＄Ｅ２９８：３－３－２節）の途中のエントリで，このエントリからはいると現在のアトリビュートを再設定しません．そのため，入力情報としてＡレジスタに消去する画面の番号を設定するだけで呼び出せます．呼び出した後は，ＰＵＴ継続フラグをクリアし，フィールドの先頭のフラグを破壊しないようにしています．

ＰＵＴＯＣＲ

ＰＵＴＯＣＲ	＄Ｅ６Ｂ４	〜	＄Ｅ６Ｃ５
ＰＵＴＯＬＦ	＄Ｅ６ＢＥ	〜	＄Ｅ６Ｃ５
ＰＵＴＯＨＭ	＄Ｅ６Ｃ６	〜	＄Ｅ６ＣＤ

機　能　機能はそれぞれのエントリで異ります．

ＰＵＴＯＣＲ　バッファポインタを現在の行の先頭へ移動します．オートＬＦが選択されていれば，改行動作をします．

ＰＵＴＯＬＦ　バッファポインタを１つ下の行へ（Ｘ座標を変えずに）移します．

ＰＵＴＯＨＭ　バッファポインタを現在のウィンドウ画面（選択されている画面）の先頭の座標へ移動します．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	.ＣＳＣＴＬ	＄Ｄ０２１	コンソール制御フラグ
	ＰＵＴＣＦＧ	＄Ｄ０２２	ＰＵＴ継続フラグ
	.ＢＰＹ	＄Ｄ０４５	バッファポインタのＹ座標
	.ＢＰＸ	＄Ｄ０４６	バッファポインタのＸ座標
	.ＭＤＲＧ	＄Ｄ０４Ｄ	ウィンドウ画面の開始行（のＹ座標）

サブルーチン　ＢＵＦＰＴＲ　＄Ｅ１ＦＣ　バッファポインタのアドレスを求める
　　　　　　　ＰＵＴＤＷ　＄Ｅ７ＤＥ　バッファポインタを１行下へ移動する

解　説　各エントリは，最終的に同じルーチン（ＰＵＴ５５０：＄Ｅ６Ｃ１～＄Ｅ６Ｃ５）に制御を移します．
各エントリによる動作は次のようになります．
ＰＵＴＯＣＲでは，ＣＲオーダを実行したことを示すために，ＰＣＲＯＲＤ（＄Ｄ０Ａ０）を０以外の値にセットします．このワークエリアは，後にＰＵＴが終了する際にＥＳＣキーによってプットウェイトするときに参照されます．バッファポインタのＸ座標を０にします．ここで，もしコンソール制御フラグ．ＣＳＣＴＬでオートＬＦが選択されていなければＰＵＴ５５０に分岐します．
ＰＵＴＯＬＦでは，ＰＵＴＤＷを用いてバッファポインタを１行下に移動したあとで，ＰＵＴ５５０に分岐します．
ＰＵＴＯＨＭでは，ウィンドウ画面の開始行のＹ座標をバッファポインタのＹ座標に代入し，バッファポインタのＸ座標を０にして，ＰＵＴ５５０に分岐します．
ＰＵＴ５５０からは，次のように動作します．
ワークエリアＰＵＴＣＦＧ（ＰＵＴ継続フラグ）をクリアした後，ＢＵＦＰＴＲルーチンで，変更されたバッファポインタのバッファアドレスを求めて，終了します．

ＰＵＴＯＳＦ

＄Ｅ６ＣＥ　～　＄Ｅ６Ｅ３

機　能　フィールドの始まりを定義します．
レジスタ　Ａ，Ｘ
ワークエリア　ＰＵＴＣＦＧ　＄Ｄ０２２　ＰＵＴ継続フラグ
　　　　　　　．．ＡＣＬ　＄Ｄ０２Ｅ　現在のアトリビュート
　　　　　　　．ＢＡＡＤ　＄Ｄ０４９～＄Ｄ０４Ａ
サブルーチン　ＰＵＴＡＴＲ　＄Ｅ６Ｅ４　フィールド内のアトリビュートバッファの設定
　　　　　　　ＰＵＴＤＡＴ　＄Ｅ７４５　共有ＲＡＭ上の出力文字列中から，１文字読み込む

解　　説　次のように動作します．

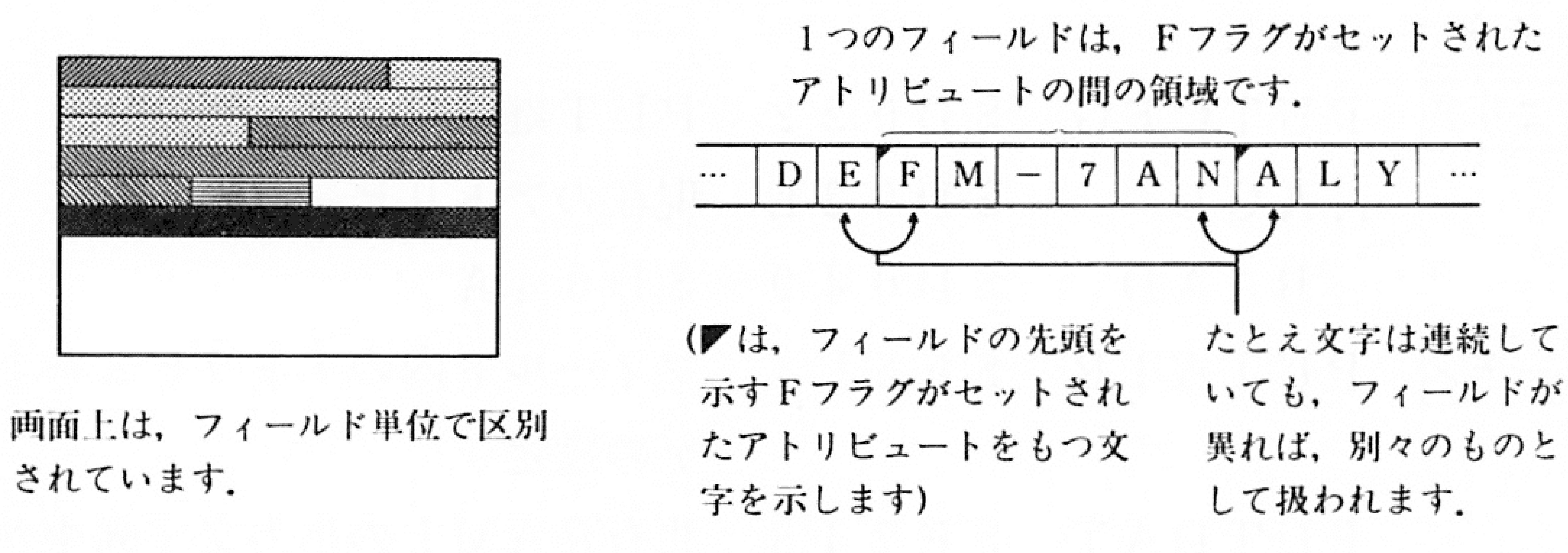
1)　ＰＵＴＤＡＴで，次の文字（この場合はオーダシーケンスのデータコード）を読み込んできます．このとき，次の文字がない場合はエラー(ＥＲＲ６２：＄Ｅ０Ｄ５）にジャンプします．読み込んできたデータコードの下位６ビットを現在のアトリビュートコードを示すワークエリア．.ＡＣＬ(＄Ｄ０２Ｅ)に格納します．

2)　ＰＵＴＡＴＲを呼び出して，現在のバッファポインタの位置から次のフィールドの手前までの部分のアトリビュートを再設定します（元のアトリビュートと新しく設定されたアトリビュートが同じときは再設定しません)．そして，バッファポインタを示す位置のアトリビュートのＦフラグ（フィールドの先頭を示すフラグ：ビット７）をセットします．最後にＰＵＴ継続フラグをクリアして終了します．

フィールドについて

ＦＭ－７，８，１１シリーズの画面（コンソール画面）は，フィールドと呼れる単位で分割されています．これは，文字列の表示処理(ＰＵＴ)，入力処理（ＧＥＴ）における単位です．
１つのフィールドというのは，Ｆフラグ（ビット７）がセットされたアトリビュートから，次のＦフラグがセットされたアトリビュートの手前までの領域を指します．ですから，画面上で連続している文字列でも，その文字列の途中でフィールドが異る場合，別々に扱われます．

図３・４・10　フィールド



フィールドの特長は，１つのフィールド内ではアトリビュートが同じであることがあげられます．よって，フィールドをアトリビュート（カラー）が同じ文字列として考えてもよいでしょう．

文字列をフィールド単位で扱うものについては，例えばGETはフィールド単位で変更された画面を転送しますし，スクリーンエディットにおいて，文字を入力していって，フィールドの境目にきたときはブザー音がなります．インサート処理でフィールド内に文字列が一杯になったときは，それ以上，文字を挿入することができなくなるなどがあり，GETでは特にフィールド単位で文字列を扱う傾向が強いようです．

PUTOSB

$E700 ～ $E730

機　　能　バッファポインタの位置を設定し，バッファポインタのバッファアドレスを設定する

レジスタ　A，B，X，Y

ワークエリア	..DMAP	$D03D～$D044	画面の情報
	.BPY	$D045	バッファポインタのY座標
	.BPX	$D046	バッファポインタのX座標
	.MDRG	$D04D～$D04F	ウィンドウ画面の情報
サブルーチン	CAXCR	$E3D5	座標の範囲をチェックする
	PUTDAT	$E745	共有RAM上の出力文字列から文字を読み込む
	GETRNG	$E975	ウィンドウ画面の範囲を示すワークエリアを設定する

解　　説　バッファポインタの位置を設定するということは，これから出力する文字列の先頭座標を決定することになります．動作は次のようになります．

1）PUTDATを用いて，共有RAM上の出力文字列から，X座標とY座標を読み込みます．そして，CAXCRでそれらの座標が画面の範囲内にあるかどうかチェックし，範囲内ならば，それを .BPXと .BPYに格納します．このとき，出力文字列がない場合や，座標が範囲外ならば，エラールーチン（ERR62：$E0D5）にジャンプします．

2）設定されたバッファポインタの位置が，現在のウィンドウ画面から出てしまう可能性があるため，バッファポインタの位置から新たなウィンドウ画面を選択します．この選択のために，画面の配置状態を示すワークエリア..DM

ＡＰを検索し，下のような条件にあてはまる画面を選択します．

画面の開始行≦バッファポインタのＹ座標＜画面の終了行(＋１)

選択された画面の番号と開始行・終了行（＋１）のＹ座標をワークエリア．ＭＤＲＧに格納し，その画面のコンソールバッファ上での範囲をＧＥＴＲＮＧによって求め，設定します．

3）最後にＰＵＴ５５０（＄Ｅ６Ｃ１：ＰＵＴＯＣＲを参照してください）にジャンプして，バッファポインタの値から，そのバッファアドレスを求め，ＰＵＴ継続フラグをクリアして終了します．

ＰＵＴＯＲＣ

＄Ｅ７３１　～　＄Ｅ７４４

機　　能	指定個数分，指定文字を表示します		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
サブルーチン	ＰＵＴＣＨＲ	＄Ｅ６７８	文字の出力
	ＰＵＴＤＡＴ	＄Ｅ７４５	共有ＲＡＭ上の出力文字列中から，文字を読み込む

解　　説　次のように動作します．

1）ＰＵＴＤＡＴを用いて，共有ＲＡＭ上の出力文字列中から文字数と文字コードを示す（オーダシーケンスの）データコードを読み込んできます．このとき，出力文字列がもうないときは，エラールーチン（ＥＲＲ６２：＄Ｅ０Ｄ５）にジャンプします．

2）読み込んだ文字数分だけ，ＰＵＴＣＨＲを用いて文字コードを出力します．

ＰＵＴＣＨＲは，文字を出力する際に必要な手続きをすべて処理していますから，Ａレジスタに文字コードを入れて，呼び出すだけでよいのです．

なお，ＲＣオーダで文字を出力するときは，オーダが割り当ててある文字コードの文字を出力することができます．例えば，Ｆ－ＢＡＳＩＣでは下記のようにします．オーダ動作を許可したままで，オーダの文字を出力するときに使用します．

ＰＲＩＮＴＣＨＲ＄(19)＋ＣＨＲ＄(１)＋ＣＨＲ＄(７)

ＰＵＴＯＢＳ

＄Ｅ７６Ｂ　～　＄Ｅ７７Ｄ

機　能	バッファポインタを１つ前に戻します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ		
ワークエリア	ＰＵＴＣＦＧ	＄Ｄ０２２	ＰＵＴ継続フラグ
	．ＢＡＡＤ	＄Ｄ０４９～＄Ｄ０４Ａ	バッファポインタのアトリビュートバッファアドレス
サブルーチン	ＢＵＦＰＴＲ	＄Ｅ１ＦＣ	バッファポインタのバッファアドレスを求める
	ＢＴＯＣ	＄Ｅ７７Ｅ	バッファポインタの位置をカーソルへ
	ＣＴＯＢ	＄Ｅ７８７	カーソルの位置をバッファポインタへ
	ＣＳＬ	＄ＥＡ６０	カーソルを１つ前に戻す

解　説　一度カーソルの位置を示すワークエリアにバッファポインタの位置を設定してから，ＣＳＬでカーソルを１つ前に戻し，それを再びバッファポインタに戻しています．

ＰＵＴＯＴＢ

＄Ｅ７９３　～　＄Ｅ７９Ｃ

機　能　ＴＡＢ動作をします．すなわち，ＴＡＢの停止位置までバッファポインタを移動させ，その移動した範囲をＳＰＡＣＥで埋めます．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	．ＢＰＸ	＄Ｄ０４６	バッファポインタのＸ座標
サブルーチン	ＰＵＴＳＰＣ	＄Ｅ６７６	空白を出力する
	ＴＡＢＣＨＫ	＄Ｅ７９Ｄ	ＴＡＢの停止位置をチェックする

解　説　次のように動作します．

１）ＰＵＴＳＰＣで，空白を１つ出力します．

２）バッファポインタのＸ座標を入力情報として，ＴＡＢＣＨＫルーチンを呼んで停止位置かどうかチェックします．停止位置でなければ，1)からの処理を繰り返します．

ＰＵＴＳＰＣは，空白を出力した後の必要な手続き（バッファポインタの更新な

ども含む）をすべて処理するので，呼び出すだけでよいのです．なお，空白すなわちＳＰＡＣＥは，Null とは異りますので注意して下さい．Null は，そこに何も書かれていないことを示しますが，ＳＰＡＣＥはそこに空白という文字が表示されているのです．画面上は同じですが，内部の処理系での扱いは異ります．このことは，サブシステムに限らず，他のデバイスでも共通しているものです．

ＴＡＢＣＨＫ

＄Ｅ７９Ｄ　～　＄Ｅ７Ｂ９

機　能	ＴＡＢの停止位置かどうかチェックします．	
レジスタ	Ａ，Ｂ，Ｘ	
入力情報	Ａレジスタ	Ｘ座標（０～７９）
出力情報	Ｚフラグ	ＴＡＢの停止位置のときクリアされ，停止位置でないときセットされる
ワークエリア	.ＣＳＣＴＬ	＄Ｄ０２１　コンソール制御フラグ
	.ＴＡＢＳＴ	＄Ｄ０２４～＄Ｄ０２Ｄ　ＴＡＢ（の停止位置）設定テーブル

解　説　Ａレジスタで与えられるＸ座標が，ＴＡＢの停止位置かどうかを判別するこのルーチンは，次のように動作します．

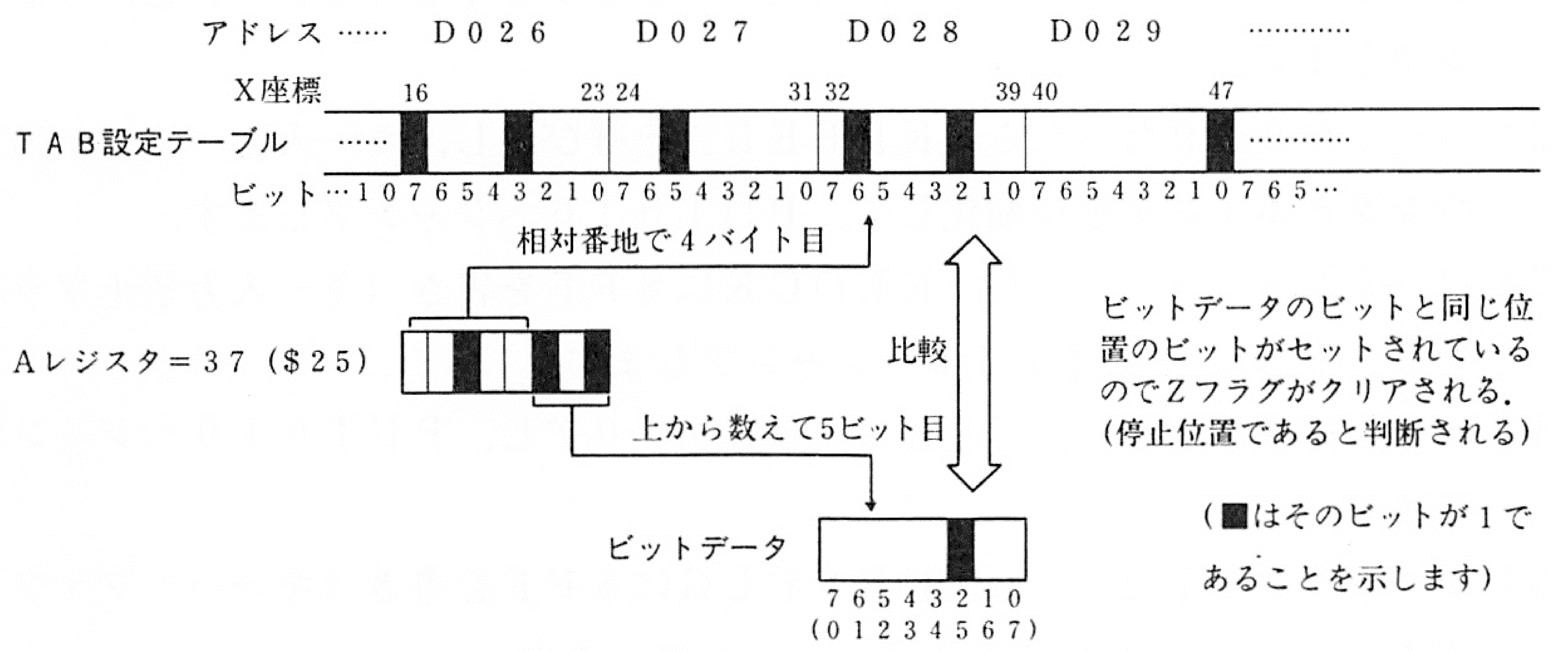
１）Ａレジスタの上位５ビットから，ＴＡＢ設定テーブルのどのバイトと比較すればよいかを決めるために，テーブル内の比較するバイトの相対番地を計算します．

２）Ａレジスタの下位３ビットから，(比較するバイト内の)どのビットを比較するかを計算します．＄Ｅ３９２～＄Ｅ３９９番地（ラベル名ＢＡＳ）に，ビットデータがあり，そのデータを読み込んできます．

３）1)で求めた相対番地によりＴＡＢ設定テーブルの比較するバイトと，2)で求めたビットデータを比較し，そのビットが１になっているかどうか調べます．１になっていればＺフラグがクリアされ，０になっていればＺフラグがセットされます．

なお，ワークエリア .ＣＳＣＴＬのビット３を見て，ＴＡＢ動作が禁止されていれば，無条件にＺフラグをクリアするようにしています．

図3・4・11　TAB設定テーブルとの比較の例



上の図は，TAB設定テーブルが図のように設定されていた場合に，Aレジスタに37を与えたときに，X座標37が停止位置かどうかを判別しています．

ESCSEQ

$E7FB　～　$E82D

機　　能　エスケープシーケンスを実行します．エスケープシーケンスは，ESCコードに続く文字コード（データコード）により，次のように機能します．

'9'（＝$39）　キー（入力）バッファを消去します．
'#'（＝$23）　キー入力を禁止（キーをロック）します．
'"'（＝$22）　キー入力禁止（キーロック）を解除します．
'g'（＝$67）　キーの先行入力を可能（キーバッファモード）にします．
'h'（＝$68）　キーの先行入力を禁止(キーバッファモードを解除)します．

レジスタ　A，X

ワークエリア　KLOCK　$D002　キー入力禁止フラグ
KBFFLG　$D003　キーバッファフラグ

サブルーチン　PUTDAT　$E745　共有RAM上の出力文字列中から文字を読み取る
KBFERA　$FD9E　キー（入力）バッファを消去する

解　　説　このルーチンは，PUTのメインルーチンから分岐してきますので，処理が終了すると再びPUTのメインルーチンへジャンプして戻ります．次のように動作します．

1）ＰＵＴＤＡＴを呼びだして，データコードを読み込んできます．このとき，文字が既に存在しないときは，ただちにＰＵＴ０１０（＄Ｅ５ＥＢ）へジャンプします．

2）コードが＄３９だったら，ＫＢＦＥＲＡを呼び出し，キー入力バッファのカウンタとポインタを初期化して，ＰＵＴ０１０へジャンプします．

3）コードが＄２３だったら，ＫＬＯＣＫに＄ＦＦを書き（キー入力禁止フラグをセットし），ＰＵＴ０１０へジャンプします．

4）コードが＄２２だったら，ＫＬＯＣＫをクリアし，ＰＵＴ０１０へジャンプします．

5）コードが＄６７だったら，ＫＢＦＦＬＧに＄ＦＦを書き（キーバッファフラグをセットし），ＰＵＴ０１０へジャンプします．

6）コードが＄６８だったら，ＫＢＦＦＬＧをクリアし，ＰＵＴ０１０へジャンプします．

なお，その他のコードがきた場合は何の動作も行いませんので，その文字は無効となり，画面上にも出力されません．ワークエリアＫＬＯＣＫとＫＢＦＦＬＧは，キー入力の際に参考にされます．３－７－１節を参照して下さい．

※カーソルの移動などのルーチンは，ＧＥＴのものを使用しています．ＧＥＴのオーダルーチン（３－４－４節）を参照して下さい．

3－4－3　GET　　　＄０４　（＄Ｅ５Ｅ０）

文字列をコンソール画面に表示した後，オペレータによって変更されたフィールドのなかで，画面最上段に位置するフィールドの内容を共有ＲＡＭに転送します．カーソルの表示位置は，文字列表示後のバッファポインタの位置になり，そこからスクリーンエディットを行います．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　容	値
Ｄ３８３	Ｎ	データバイト数	今回転送する文字列のバイト数（Ｎ≦１２４）
Ｄ３８４ ～ Ｄ383＋Ｎ	String	出力文字列	出力する文字列（オーダを含む）

共有ＲＡＭ上の復帰情報

アドレス		内　容	値
Ｄ３８３	Ｋ	終了キーコード	＄００：ＲＥＴＵＲＮ ＄０１：ＣＴＲＬ＋ＣまたはＣＴＲＬ＋Ｘ ＄０２：ＣＬＳ
Ｄ３８４	Ｎ	今回転送する，変更フィールドのバイト数（Ｎ≦１２０）	変更フィールドのないときはＮ＝０，変更フィールドが空文字列のときは，Ｎ＝３（ＳＢＡ，Ｘ，Ｙのみ）
Ｄ３８５	ＳＢＡ	Set Buffer Address（オーダ）コード	＄１２
Ｄ３８６	Ｘ	変更フィールドの先頭（の文字）のＸ座標	０～７９
Ｄ３８７	Ｙ	変更フィールドの先頭（の文字）のＹ座標	０～２４
Ｄ３８８ ～ Ｄ384＋Ｎ	ＴＥＸＴ	フィールド中の制御コードを除く文字列	変更フィールドの文字列

補助ワークエリア

$D0A0 ~$D0A4	PCRORD~	PUT（3－4－1節）参照
$D0A5	GTFLD	カーソルの文字列単位での移動のときに，フィールドの区切りに来たかどうかを示すフラグ
$D0A6	GTSCRL	インサートモードのときのための，スクロールを行ったかどうか示すフラグ(行わないとき$00，行ったとき$FF)

解　説　GETの処理は，次のように大きく3つに分かれます．

1）文字列表示処理

共有RAM上に渡された出力文字列の表示は，PUTと同じ処理なので，PUTルーチンで行っています．このため，GETのエントリはPUTルーチンのすぐ近くにあり，PUTのルーチンに入り，表示処理が終った段階でPUTルーチンを抜けだしてきます．詳細はPUT（3－4－1節）を参照して下さい．

2）スクリーンエディット処理

PUTルーチンから制御をGET000（$E930）に移します．ここからがGETの処理系が始まります．GETのメインルーチン（GET100：$E941～$E975）を中心に，各オーダの処理や文字の入力･表示を行います．各オーダへは，ジャンプテーブルを用います．

3）変更フィールドの出力処理

スクリーンエディットで変更されたフィールドを，共有RAMへ出力します．CLS，CTRL－C・X，RETURNキーを押したときに，この処理部へ制御が移ることになります．この処理は，GETCルーチンを用いて行っていますのでGETCルーチンで説明します．

このようにスクリーンエディット処理が，GET自体の処理であり，他の処理は，PUT・GETCのコマンドルーチンと兼ねています．スクリーンエディットを行っているGET000，GET100を次に説明します．

ＧＥＴ０００

ＧＥＴ０００　　　＄Ｅ９３０　～　＄Ｅ９７４

ＧＥＴ１００　　　＄Ｅ９４１

機　能　スクリーンエディット（オペレータによるキー入力操作）処理を統轄します．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

ワークエリア

..ＡＢＲＴ	＄Ｄ０００	キャンセルフラグ
.ＣＵＲＴＭ	＄Ｄ０１Ｄ	カーソルタイマ
ＰＵＴＣＦＧ	＄Ｄ０２２	ＰＵＴ継続フラグ
ＧＴＴＯＰＡ	＄Ｄ０５４～＄Ｄ０５５	ウィンドウ画面のアトリビュートバッファでの先頭アドレス
ＧＴＢＴＭＡ	＄Ｄ０５６～＄Ｄ０５７	ウィンドウ画面のアトリビュートバッファでの終了アドレス＋１

サブルーチン

ＣＢＡＤＲ	＄Ｅ３Ｃ３	カーソルのバッファアドレスを求める
ＧＥＴＤＴ	＄Ｅ９８Ａ	入力された文字の格納・表示
ＧＥＴＤＥＬ	＄ＥＢＥ８	ＤＥＬオーダの実行
ＣＵＲＳＥＴ	＄ＥＤＥ８	カーソルの位置の設定（継承）
ＣＵＲＳＣＬ	＄ＥＤＥＤ	カーソルのクリア
ＣＵＲＧＯ	＄ＥＥ０７	カーソルを点滅させる
ＫＥＹＧＥＴ	＄ＦＤ６Ｅ	キー入力

解　説　ＧＥＴ０００～ＧＥＴ１００の間の処理で，初期化を行っています．ＧＥＴ１００以降がメインルーチンで，キー入力の処理を行っています．それぞれの処理は次のようになります．

１）初期化処理　ＧＥＴ０００～

ＰＵＴ継続フラグＰＵＴＣＦＧをクリアして，バッファポインタ（カーソル）がフィールドの先頭にきても，フィールドの先頭を示すフラグ（Ｆフラグ）を消すことがないようにしています．その後，現在のウィンドウ画面の始めから終りまでのアトリビュートのＭフラグ（変更フラグ：ビット６）をクリアして，まだオペレータによる操作がないことを示させます（後で，操作が行われた所のアトリビュートのＭフラグはセットされます）．また，ＣＵＲＳＥＴを用いて，カーソ

ルを消去し，ＰＵＴルーチン終了時のバッファポインタのワークエリアの内容を，カーソルのワークエリアに複写しておきます．この後，次の処理に入ります．

２）キー入力処理　ＧＥＴ１００～

カーソルの座標から，カーソルのバッファアドレスとＶＲＡＭ上のアドレスを求めた（ＣＢＡＤＲを用います）後，点滅を開始させます．点滅が行われている間，キー入力を行います．入力後は，カーソルの点滅を終了させ，画面上からカーソルを消去します．このとき，キャンセル割込みがメインシステムから要求された場合（..ＡＢＲＴ≠０）は，コマンド解釈ルーチン（ＣＭＤＣＴＬ：＄Ｅ０３４）に強制的に戻ります．キャンセル割込みがないときは，キー入力されたキーコードによって，各ルーチンを呼び出します．呼び出しは下のようになります．呼び出した後は，ＧＥＴ１００からの処理を繰り返します．

ｉ）普通のキーの場合

ＧＥＴＤＴを呼び出して，キー入力されたデータをコンソールバッファに格納し，ＶＲＡＭ上に文字を表示させます．

ii）オーダのキーの場合

各オーダへのジャンプベクトル（ＧＥＴＪＭＰ：＄ＥＣＡＣ～＄ＥＣＥＢに格納されています）によって，各オーダへのサブルーチンにジャンプします．ただし，ＤＥＬキーの場合は，コードが＄７Ｆなので特別に判別して，ＧＥＴＤＥＬ（＄ＥＢＥ８）を呼びだします．

各キーの処理が終った後は，ＧＥＴ１００からの処理を再び繰り返します．各オーダへは，サブルーチンとして呼びだしますので，エディットを終了させるキーが押されたときは，各オーダルーチンでスタック上に退避されたリターンアドレスを放棄して，ＧＥＴＣに制御を移すようにしています．各オーダのサブルーチンのアドレスは，図３・４・13のとおりです．

図３・４・12　GETの３つの処理

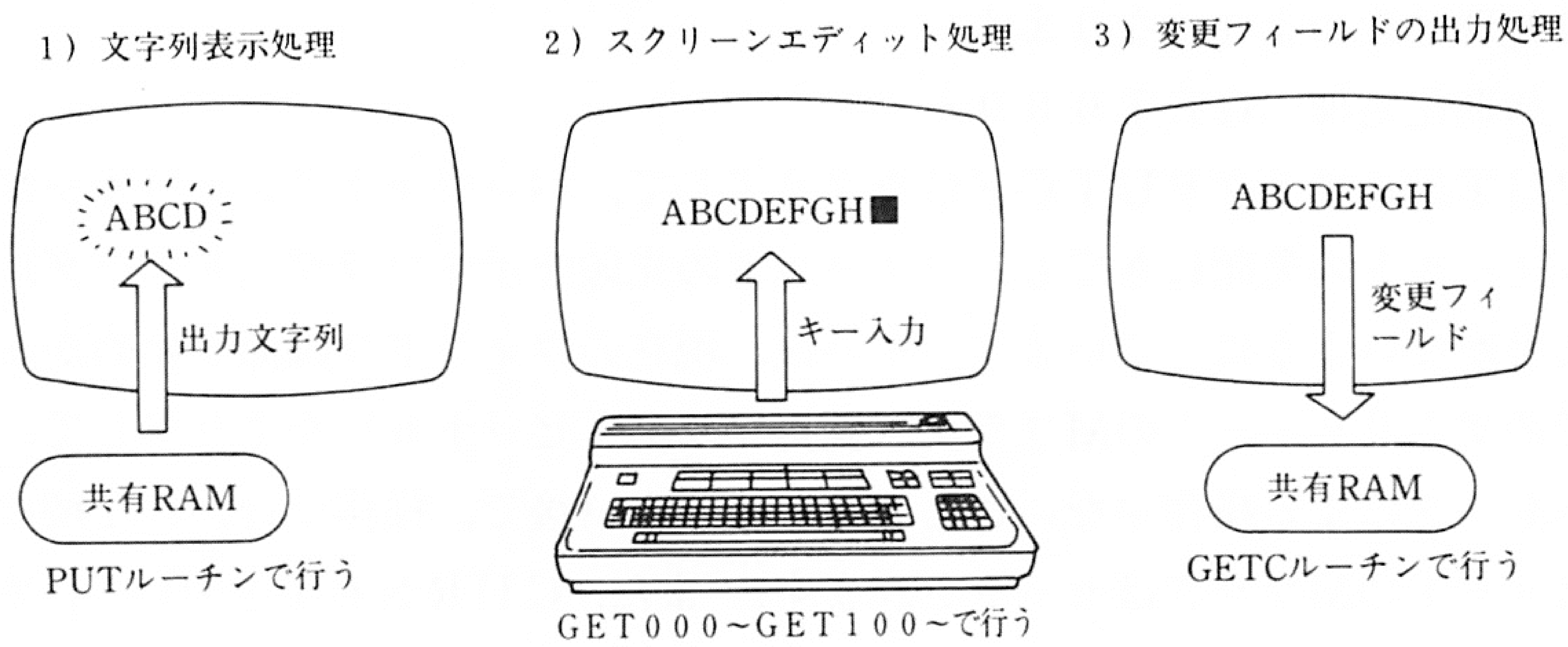


図3・4・13 GETのオーダのサブルーチン

オーダコード	アドレス	ラベル名	オーダコード	アドレス	ラベル名
$00~$01	$E989	GETNOP	$0E~$10	$E989	GETNOP
$02	$EBA4	CSLW	$11	$EB21	GETDUP
$03	$EC3E	GETCAN	$12	$EAE0	INSERT
$04	$E989	GETNOP	$13~$17	$E989	GETNOP
$05	$EAEC	ELKEY	$18	$EC3E	GETCAN
$06	$EB56	CSRW	$19	$EAB8	GETBFD
$07	$E6A8	GETBEL	$1A	$EAB0	GETFFD
$08	$EBDC	GETBS	$1B	$E989	GETNOP
$09	$EB2B	TAB	$1C	$EA14	CSRL
$0A	$E989	GETNOP	$1D	$EA1F	CSLL
$0B	$EA94	CSHOME	$1E	$EA29	CSUL
$0C	$EAD2	GETCLR	$1F	$EA37	CSDL
$0D	$EC41	GETCR	$7F	$EBE8	GETDEL

※GETNOPは，何もしないでただリターン（RTS）するだけです。

GETDT

GETDT　　$E98A ～ $E9D1
GETDT1　 $E99D

機　能	キー入力された文字を表示し，バッファへ格納します．		
レジスタ	A，B，X，Y，U		
入力情報	Aレジスタ	キー入力された文字（コード）	
ワークエリア	..INS	$D033	インサートフラグ
	.BPY	$D045	バッファポインタY座標
	.BPX	$D046	バッファポインタX座標
	.MDRG+2	$D04F	現在のウィンドウ画面の種類
	.CY	$D058~$D059	カーソルのXY座標
	.CBAD	$D05A~$D05B	カーソルのキャラクタバッファアドレス
	.CAAD	$D05C~$D05D	カーソルのアトリビュートバッファアドレス
	GTSCRL	$D0A6	スクロール実行フラグ

サブルーチン

BUFPTR	$E1FC	バッファアドレスの設定
CBADR	$E3C3	カーソルのバッファアドレスを求める
CDSPC	$E518	キャラクタをカーソル位置へ表示
CTOB	$E787	カーソルの位置をバッファポインタへ
SCROLL	$E82E	スクロール画面をスクロールアップ
GETINS	$E9D2	インサートモードのときの処理
CSR	$EA44	カーソルを１つ進める

解　説　このルーチンの処理は，インサートモードであるかどうかによって異ります．インサートモードのときは，GETINSルーチンへ分岐します．インサートモードでないときは，GETDT1から始まるルーチンで処理を行います．また，GETDT1は，GETINSルーチンからサブルーチンとして呼ばれており，そのときの入力情報は次のようになります．

Aレジスタ　　キー入力された文字　　Bレジスタ　　アトリビュートコード
Xレジスタ　　文字を格納するアドレス（キャラクタバッファ内）
Yレジスタ　　アトリビュートを格納するアドレス(アトリビュートバッファ内)

このルーチンの動作は，次のようになります．

1）モード判別処理

カーソルのバッファアドレスのワークエリア（.CBADおよび.CAAD)から，文字を格納するコンソールバッファ内のアドレスを取り出し，アトリビュートバッファからアトリビュートコードを読み込みます．そのアトリビュートの保護フラグ（Pフラグ：ビット4）を見て，保護フィールドのときは，ブザーを鳴らして($D403をRead)終了します．フィールドでないときは，インサートモードならGETINSへ分岐し，インサートモードでないならGETDT1で次の処理を行います．

2）文字の表示・格納処理　　GEDT1～

キー入力された文字コードを，キャラクタバッファに格納します．また，アトリビュートコードは変更フラグ（Mフラグ：ビット6）をセットされてアトリビュートバッファへ格納されます．後で共有RAM上に変更されたフィールドを送り込むときに，この変更フラグが参照されます．次に，CDSPCでキャラクタのフォントをVRAMに書き込み，文字を画面に表示させます．

3）カーソルを進める処理

CSRルーチンを用いてカーソルを１つ進めます．もし，スクロールが必要ならばSCROLLルーチンを呼んで画面を１行スクロールアップさせます．スク

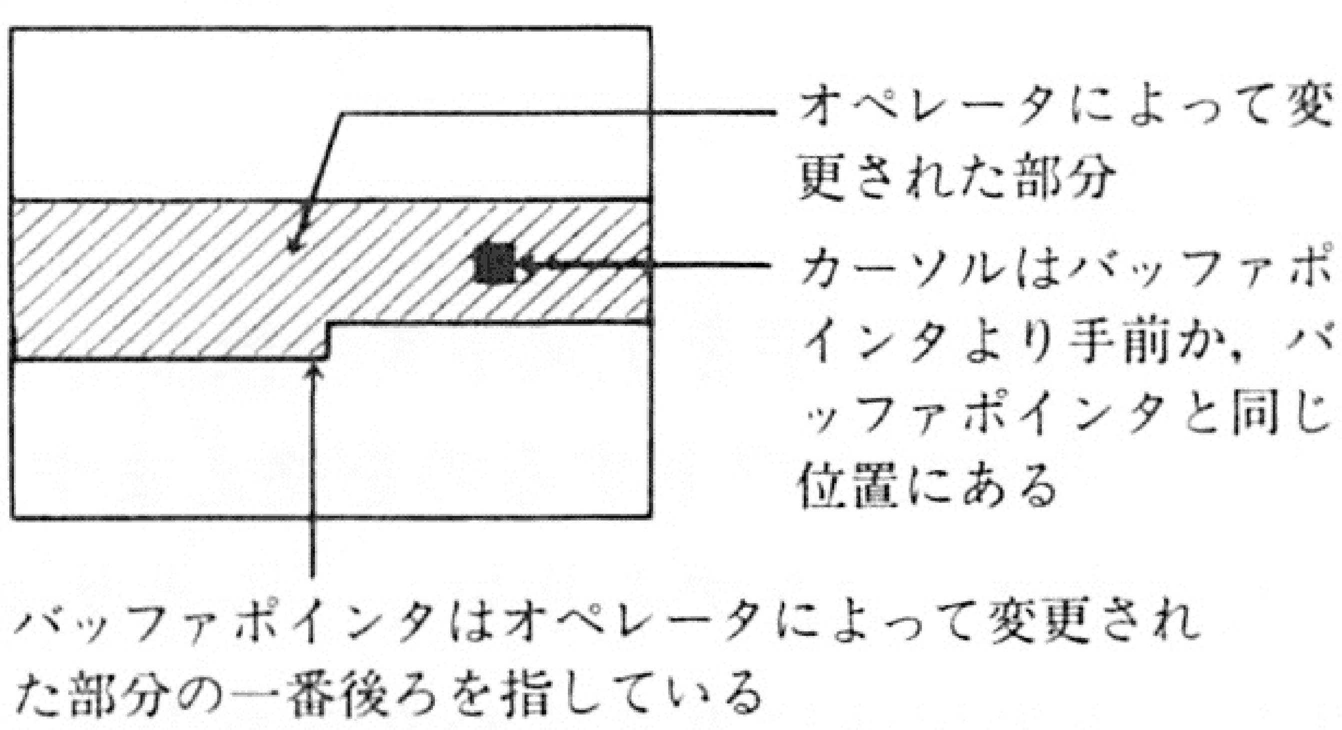
ロールした場合は，スクロールしたことをワークエリアGTSCRLに設定します．これは，GETINSルーチン内でこのルーチンを呼んでいるために，スクロールしたことをGETINSに知らせる目的で行っています．

SCROLLルーチンでは，バッファポインタのY座標を更新しているため，カーソルの座標をバッファポインタから読み取ります．またバッファポインタに対しても，そのバッファアドレスを更新させています．

ページ画面であったり，スクロールをする必要のない場合は，カーソルのバッファアドレスを求めます．そして，カーソルのある位置がフィールドの先頭ならばブザーを鳴らします．なお，バッファポインタの位置がカーソルの位置よりも手前にあるときは，カーソルの位置をバッファポインタに複写して終了します．

GET処理系では，バッファポインタは，オペレータによって変更された部分の最後を示しています．そのために，カーソルの位置がバッファポインタの位置より後にくることがあれば，バッファポインタの位置を更新して（カーソルの位置にあわせて）いるのです．

図3・4・14 バッファポインタのGET処理系での役割



GETINS

$E9D2 ～ $EA13

機　能	挿入処理を行い，キー入力された文字を格納・表示します．	
レジスタ	A，B，X，Y，U	
入力情報	Aレジスタ	キー入力された文字（のコード）
	Bレジスタ	アトリビュートコード
	Xレジスタ	文字を格納するアドレス（キャラクタバッファ内）
	Yレジスタ	アトリビュートを格納するアドレス（アトリビュートバッファ内）

ワークエリア	.MDRG	$D04D	ウィンドウ画面の開始行のY座標
	.CY	$D058	カーソルのY座標
	.CX	$D059	カーソルのX座標
	.CBAD	$D05A～$D05B	カーソルのキャラクタバッファアドレス
	.CAAD	$D05C～$D05D	カーソルのアトリビュートバッファアドレス
	GTSCRL	$D0A6	スクロール実行フラグ
サブルーチン	CBADR	$E3C3	カーソルのバッファアドレスを求める
	GETDT1	$E99D	入力文字を表示・バッファへ格納
	CSR	$EA44	カーソルを1文字分進める

解　説　GETDTルーチンの途中から分岐してくるこのルーチンは，次のように動作します．

1）Null コードの探索

現在のカーソルの位置から，次のフィールドまでの間にNull コードがあるかどうか探索します．Null コードが1つもなければ，挿入する場合がないのでブザーを鳴らして終了します．

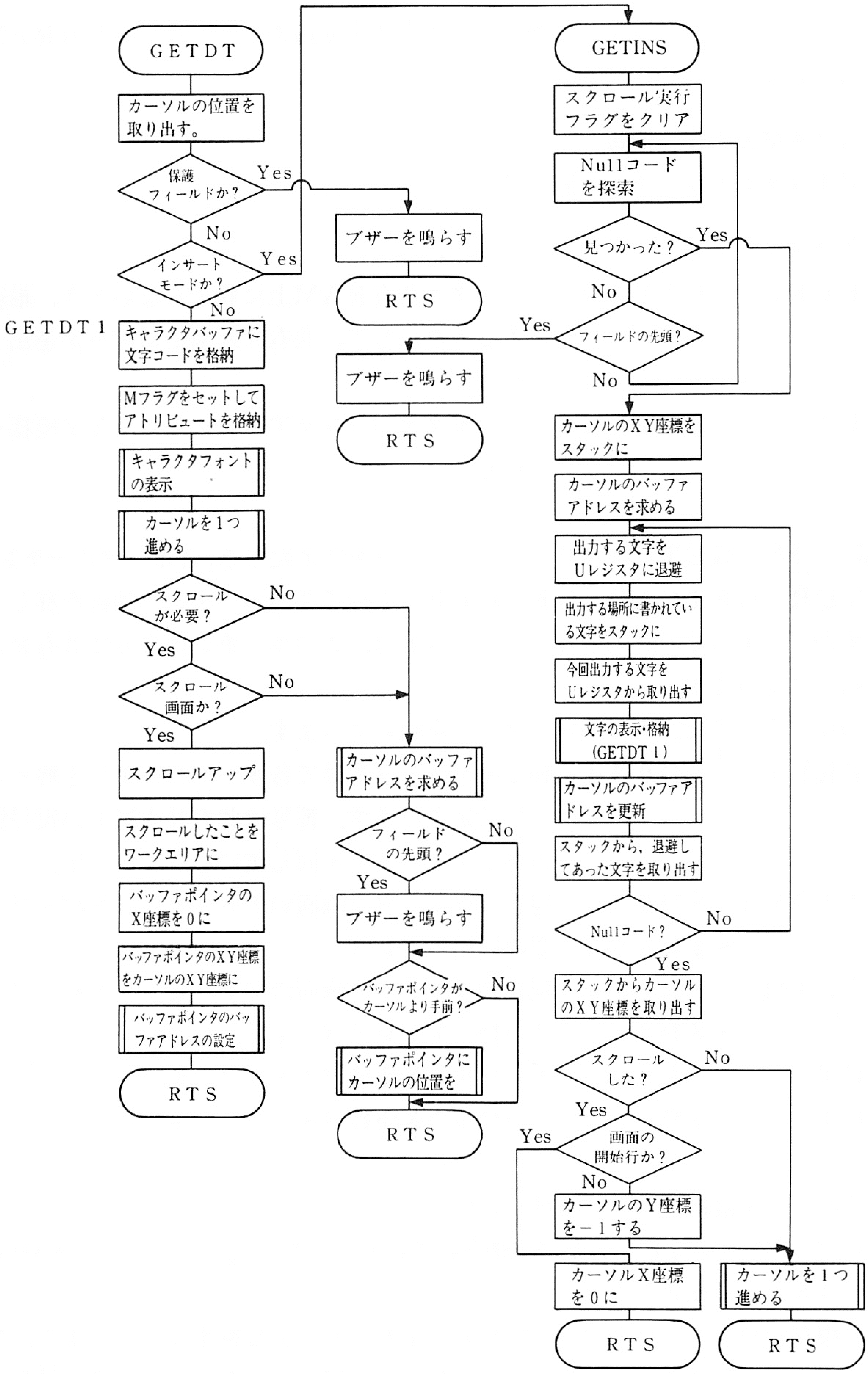
2）1文字分後ろへずらす

現在のカーソルの位置以降の文字を1文字ずつ後ろへずらします．ずらす文字がNull コードであれば，そこでシフトするのを終えます．シフトする際には，シフトする場所にあった文字コードとアトリビュートをスタック上に退避した後，GETDT1で，新たにその場所に書き込む文字を表示・格納させ，次の回にスタック上に退避してあった文字を書き込んでいくという方法をとっています．

3）カーソルの設定

シフトが終った後，挿入する場所のカーソル位置を設定し，シフトしているとき，スクロールしなかったのであれば，CSRでカーソルを1つ進めて終了します．もし，スクロールしたのであれば，1行上に挿入場所が移動していますから，カーソルを1つ上にしてから，CSRを呼び出します．ただし，画面の開始行にあった場合は，行の先頭へカーソルが持って行かれます．

図3・4・15　GETDT，GETINSの処理



ＧＥＴＣ　　　　　＄０５　（＄ＥＣＡ６）

オペレータによって修正されたフィールドで未転送のフィールドを共有ＲＡＭ上に設定します．

共有ＲＡＭ上の復帰情報

ＧＥＴコマンドルーチンと同じです．

主なサブルーチン

＄Ｅ０Ｆ５	ＣＯＮＴＯ	データが共有ＲＡＭ上に収まらないとき，継続することを示して，共有ＲＡＭ上にデータを出力させる
＄ＥＡ７９	ＧＥＴＥＮＣ	キャラクタバッファアドレスから，ＸＹ座標を求める

解　説　ＧＥＴでスクリーンエディットを終えたとき，各オーダルーチン（ＧＥＴＣＲ，ＧＥＴＣＡＮ，ＧＥＴＣＬＲ）を経てこのルーチンへ制御を移してきますから，すべての修正されたフィールドは，このルーチンによって共有ＲＡＭへ送られています．

このルーチンでは，次のワークエリアを用いています．

．ＥＫＥＹ	＄Ｄ０３４	終了キーコード番号で各オーダルーチンを終えるときに設定されます．番号は共有ＲＡＭ上の復帰情報の終了キーコードと同じ値になっています．
ＧＴＴＯＰＡ	＄Ｄ０５４〜＄Ｄ０５５	現在のウィンドウ画面のアトリビュートバッファ上での開始アドレス
ＧＴＢＴＭＡ	＄Ｄ０５６〜＄Ｄ０５７	現在のウィンドウ画面のアトリビュートバッファ上での終了アドレス（＋１）
．ＣＹ	＄Ｄ０５８	カーソルのＹ座標（キャラクタ単位）
．ＣＸ	＄Ｄ０５９	カーソルのＸ座標（キャラクタ単位）

このルーチンは次のように処理します．

1）ワークエリア．ＥＫＥＹに格納されている終了キーコードを，共有ＲＡＭ（＄Ｄ３８３）に転送します．

2）現在のウィンドウ画面の先頭からアトリビュートを検索していきます．フィールドの先頭がくるたびに，そのバッファアドレスをＵレジスタに保存しておきます．そして，アトリビュートコードの変更フラグ（Ｍフラグ：ビット

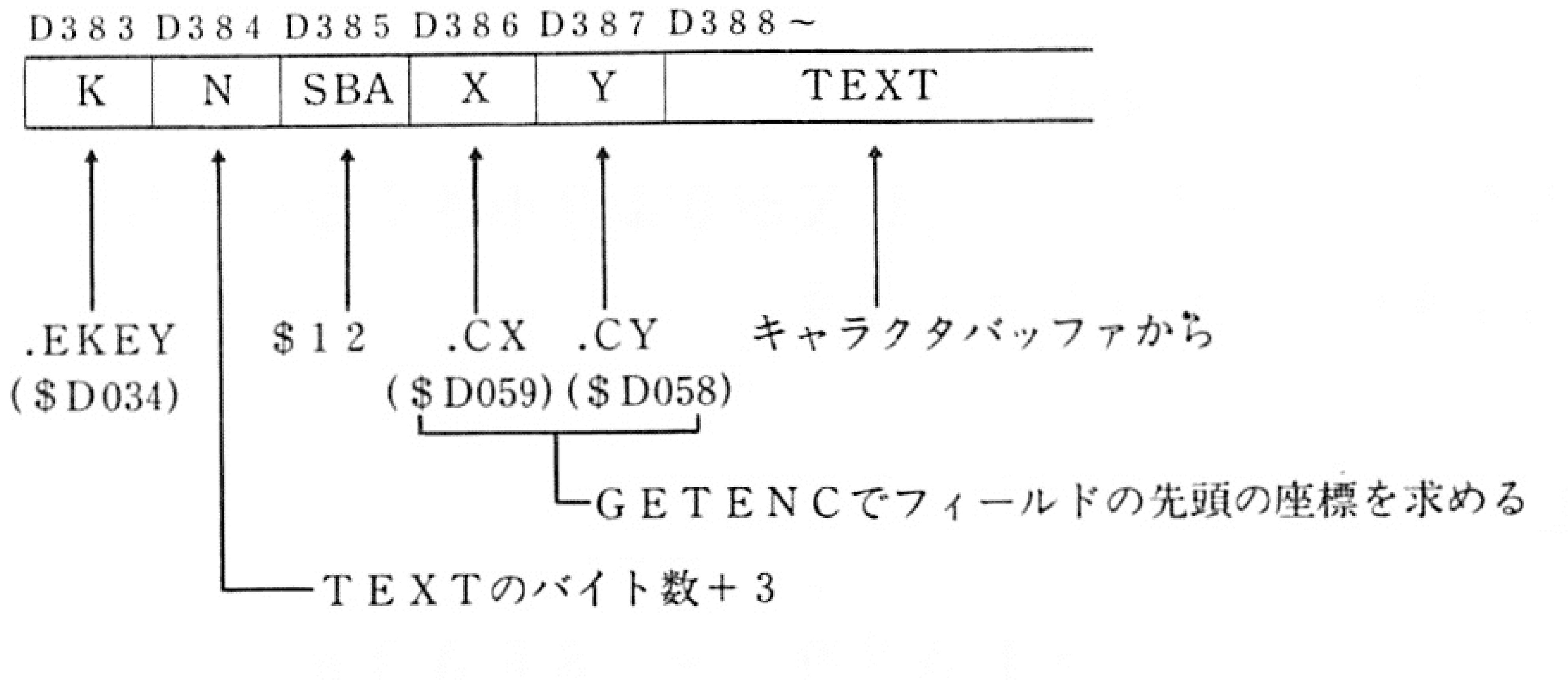
６）がセットされていれば，次の処理へ制御を移します．もし，ウィンドウ画面の終端まで，変更フラグがセットされたアトリビュートがなければ，未転送の変更されたフィールドはないとして，共有ＲＡＭ上の転送バイト数(Ｎ：＄Ｄ３８４）を０にして終了します．

３）Ｍフラグがセットされたアトリビュートが検索された位置から次のフィールドまでの間のアトリビュートコードのＭフラグをクリアしていきます．これは，次にこのルーチンを呼んだときに，同じ場所（フィールド）が検索されないようにするためです．

４）今回転送するフィールドの先頭のキャラクタバッファアドレスとアトリビュートバッファアドレスを，Ｕレジスタに保存してあった（アトリビュート）バッファアドレスから求めます．また，そのアドレスからＸＹ座標をＧＥＴＥＮＣルーチンを使って求め，ＳＢＡオーダコード（＄１２）と共に共有ＲＡＭ上（＄Ｄ３８５～＄Ｄ３８７）に転送します．ＳＢＡオーダコードと共に転送するのは，ユーザが同じ場所にエコーバックをするのに便利なように，という配慮によるものです．

５）フィールドの先頭のキャラクタバッファアドレスから転送するフィールドのキャラクタコードを共有ＲＡＭ上（＄Ｄ３８８～）に転送していきます．１文字転送するごとに，転送バイト数Ｎ（＄Ｄ３８４：初期値は３）を＋１していきます．共有ＲＡＭが一杯になったときは，ＣＯＮＴＯルーチンで一度メインシステムがデータを受け取るのを待ってから，再び共有ＲＡＭ上(＄Ｄ３８５～）にキャラクタを転送します．なお，一度転送した後は，転送バイト数Ｎをクリアをしてから転送を行っていきます．フィールド内のキャラクタがすべて転送されれば終了します．

図３・４・16　共有ＲＡＭ上へのデータ転送



３－４－４　ＧＥＴのオーダサブルーチン

ＣＳＲＬ

＄ＥＡ１４　～　＄ＥＡ１Ｅ

機　　能	カーソルを１つ右に移動します．		
レジスタ	Ａ		
ワークエリア	..ＬＮＣ	＄Ｄ０３６	１行の桁数
	.ＣＸ	＄Ｄ０５９	カーソルのＸ座標

解　　説　このルーチンおよび後に続く５つのサブルーチンでは，ＧＥＴのメインルーチン（ＧＥＴ１００：＄Ｅ９４１）がカーソルの座標からそのバッファアドレスを求めているおかげで，ただ，カーソルの座標（を示すワークエリア）を更新しているだけです．

動作は次のようになります．

１）　.ＣＸを＋１します．このとき .ＣＸが..ＬＮＣと等しくなるか，..ＬＮＣより大きくなったら，.ＣＸを０にします．

ＣＳＬＬ

＄ＥＡ１Ｆ　～　＄ＥＡ２８

機　　能	カーソルを１つ左に移動します．		
レジスタ	Ａ		
ワークエリア	..ＬＮＣ	＄Ｄ０３６	１行の桁数
	.ＣＸ	＄Ｄ０５９	カーソルのＸ座標

解　　説　次のように動作します．

１）　.ＣＸを－１します．もし .ＣＸが０より小さくなったら，.ＣＸに..ＬＮＣから１を引いた値を代入します．

ＣＳＵＬ

＄ＥＡ２９　～　＄ＥＡ３６

機　　能　カーソルを１つ上に移動します．

レジスタ	A		
ワークエリア	.MDRG	$D04D	ウィンドウ画面の開始行(のY座標)
	.MDRG+1	$D04E	ウィンドウ画面の終了行の次の行(のY座標)
	.CY	$D058	カーソルのY座標

解説 次のように動作します.

1) .CYを−1します. もし, .CYが .MDRGより小さくなったら, .MDRG+1の値から1を引いたのを, .CYに代入します.

このように,範囲のチェックを現在のウィンドウ画面の範囲で行っていますので,カーソルを動かして, ウィンドウ画面以外の画面に抜け出ることはできません.

CSDL

$EA37 ~ $EA43

機能	カーソルを1つ下に移動します.		
レジスタ	A		
ワークエリア	.MDRG	$D04D	ウィンドウ画面の開始行
	.MDRG+1	$D04E	ウィンドウ画面の終了行の次の行
	.CY	$D058	カーソルのY座標

解説 次のように動作します.

1) .CYを+1します. もし, .CYが .MDRG+1より大きいか, 等しいときは, .MDRGの値を .CYに代入します.

CSR

$EA44 ~ $EA5F

機能	カーソルを1つ進めます.		
レジスタ	A		
出力情報	Cフラグ	カーソルが画面の範囲を越えたときセットされます	
ワークエリア	..LNC	$D036	1行の桁数
	.MDRG	$D04D	ウィンドウ画面の開始行
	.MDRG+1	$D04E	ウィンドウ画面の終了行の次の行

.ＣＹ　　　　＄Ｄ０５８　カーソルのＹ座標
.ＣＸ　　　　＄Ｄ０５９　カーソルのＸ座標

解　説　次のように動作します.

1）.ＣＸを＋１し，それが..ＬＮＣより大きいか等しいときは，.ＣＸを０にし，.ＣＹを＋１します.

2）.ＣＹが.ＭＤＲＧ＋１より大きいか，等しいときは，.ＭＤＲＧの値を.ＣＹに代入し，このときＣフラグをセットします.

このように，このルーチンはＣＳＲＬ（＄ＥＡ１４）と異り，カーソルが画面の右端にあれば，次の行の先頭へもっていきます．また，ウィンドウ画面の終端にあれば，ウィンドウ画面の先頭へもっていきます.

ＣＳＬ

＄ＥＡ６０　～　＄ＥＡ７８

機　能	カーソルを１つ前に戻します.		
レジスタ	Ａ		
出力情報	Ｃフラグ	画面の範囲を越えたときセットされます	
ワークエリア	..ＬＮＣ	＄Ｄ０３６	１行の桁数
	.ＭＤＲＧ	＄Ｄ０４Ｄ	ウィンドウ画面の開始行
	.ＭＤＲＧ＋１	＄Ｄ０４Ｅ	ウィンドウ画面の終了行の次の行
	.ＣＹ	＄Ｄ０５８	カーソルのＹ座標
	.ＣＸ	＄Ｄ０５９	カーソルのＸ座標

解　説　ＣＳＲと逆の動作をします.

1）.ＣＸを－１し，それが０より小さくなったら，..ＬＮＣから１引いた値を代入し，.ＣＹを－１します.

2）.ＣＹが.ＭＤＲＧより小さくなったら，.ＭＤＲＧ＋１から１引いた値を.ＣＹに代入し，Ｃフラグをセットします.

つまり，カーソルが画面の左端にあれば，前の行の右端へ，ウィンドウ画面の先頭にあれば，ウィンドウ画面の終端へ，カーソルをもっていきます.

ＣＳＨＯＭＥ

ＣＳＨＯＭＥ　　＄ＥＡ９４　～　＄ＥＡＡＦ
ＣＳＨＯＭ０　　＄ＥＡ９７

機　能	カーソルを画面の先頭に移動します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ		
入力情報	Ｙレジスタ	フィールドの検索をはじめるアトリビュートバッファアドレス（ＣＳＨＯＭ０のみ）	
ワークエリア	ＧＴＴＯＰＡ	＄Ｄ０５４～＄Ｄ０５５	ウィンドウ画面の先頭のアトリビュートバッファアドレス
	ＧＴＢＴＭＡ	＄Ｄ０５６～＄Ｄ０５７	ウィンドウ画面の終端のアトリビュートバッファアドレス
サブルーチン	ＧＥＴＥＮＡ	＄ＥＡ７Ｆ	アトリビュートバッファアドレスから，Ｘ座標，Ｙ座標を求め，カーソルの座標を示すワークエリア．ＣＸ，．ＣＹに格納する．

解　説　ＣＳＨＯＭ０は，与えられたアトリビュートバッファアドレスから，その次のフィールドの先頭のアトリビュートバッファアドレスを求め，それでカーソルの座標を示すワークエリアを設定するものです．ＣＳＨＯＭＥは，ＣＳＨＯＭ０に与えるアトリビュートバッファアドレスの初期値として，ＧＴＴＯＰＡの値を用いています．

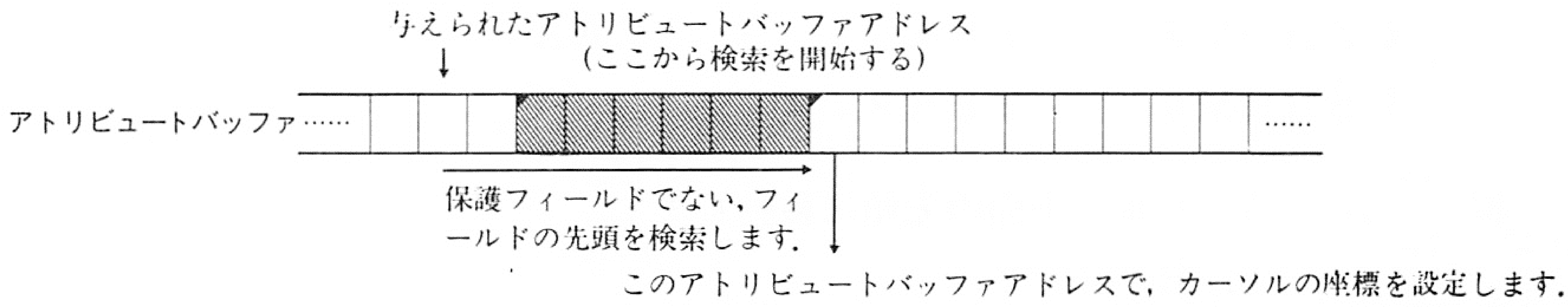
ＣＳＨＯＭ０からは，次のように動作します．

1）与えられたアトリビュートバッファアドレス以降のアトリビュートバッファのアトリビュートを検索していき，フィールドの先頭で，かつ，保護フィールドでないアトリビュートを見つけます．

2）現在のウィンドウ画面の終端まで，1)の条件に適合するようなアトリビュートがない場合は，ウィンドウ画面の先頭のアトリビュートバッファアドレスを選択します．

3）1)および2)の操作から得られたアトリビュートバッファアドレスからＧＥＴＥＮＡを用いて，カーソルの座標を設定します．

図3・4・17　CSHOM0の動作例



※斜線の部分は，保護フィールドを示します．◤はフィールドの先頭を示します．

GETFFD

$EAB0　～　$EAB7

機　　能	カーソルを次のフィールドの先頭へ移動させます．		
レジスタ	A，B，X，Y		
サブルーチン	CBADR	$E3C3	カーソルのバッファアドレスを求める
	CSR	$EA44	カーソルを１つ先に進める
	CSHOM0	$EA97	指定されたアトリビュートバッファアドレスから，次のフィールドの先頭にカーソルを設定します．

解　　説　処理は，CSHOM0で行っています．ただし，カーソルをあらかじめ１つ進めておいて(CSR)，そのアトリビュートバッファアドレスをCBADRを用いてYレジスタに設定してから呼び出しています．その理由は，カーソルがフィールドの先頭にあるとき，そのままCSHOM0を呼び出すと，その位置の座標をカーソルの座標として設定するからです．

GETBFD

GETBFD　　$EAB8　～　$EAD1
GETBHM　　$EABE

機　　能	カーソルを前のフィールドの先頭へ移動します．		
レジスタ	A，B，X，Y		
入力情報	Yレジスタ	フィールドの検索を開始するアトリビュートバッファアドレス（GETBHMのみ）	
ワークエリア	GTTOPA	$D054～$D055	ウィンドウ画面の先頭のアトリビュートバッファアドレス

サブルーチン CBADR $E3C3 カーソルのバッファアドレスを求める
CSL $EA60 カーソルを1つ前に戻す
GETENA $EA7F アトリビュートバッファアドレスから，X座標，Y座標を求め，それをカーソルの座標を示すワークエリア .CX，.CYに設定する．

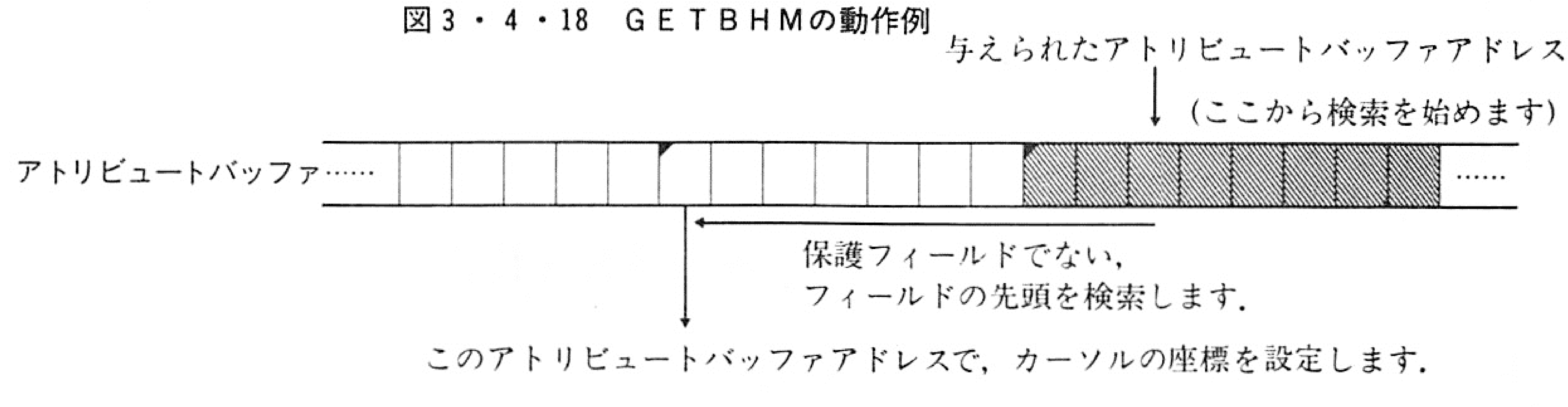
解　説 GETBHMからは，CSHOMと逆の処理を行います．Yレジスタで与えられたアトリビュートバッファアドレスから，その前のフィールドの先頭のアトリビュートバッファアドレスを求め，それでカーソルの座標を示すワークエリアを設定するものです．GETBHMからは，次のように動作します．

1）与えられたアトリビュートバッファアドレス以前のアトリビュートバッファのアトリビュートを検索していき，フィールドの先頭でかつ保護フィールドでない（Fフラグ：ビット7がセットされていて，Pフラグ：ビット4がクリアされている）アトリビュートを見つけます．

2）現在のウィンドウ画面の先頭まで，1)の条件に適合するようなアトリビュートがない場合は，ウィンドウ画面の先頭のアトリビュートバッファアドレスを選択します．

3）1)および2)の操作から得られたアトリビュートバッファアドレスから，GETENAを用いて，カーソルの座標を設定します．

図3・4・18 GETBHMの動作例



※斜線の部分は，保護フィールドを示し，◤はフィールドの先頭を示します．

GETBHMでは，与えられたアドリビュートバッファアドレスから検索しますので，それがフィールドの先頭である場合を考えて，GETBFDでは，あらかじめカーソルを1つ前に戻して(CSL)，そのアトリビュートバッファアドレスをCBADRを用いてYレジスタに設定してから，GETBHMに制御を移しています．

ＧＥＴＣＬＲ

＄ＥＡＤ２　～　＄ＥＡＤＦ

機　能	ウィンドウ画面を消去後，スクリーンエディットも終結させます．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	．ＢＰＹ	＄Ｄ０４５	バッファポインタのＹ座標
	．ＢＰＸ	＄Ｄ０４６	バッファポインタのＸ座標
	．MDRG＋２	＄Ｄ０４Ｆ	ウィンドウ画面の画面番号
	．ＣＹ	＄Ｄ０５８	カーソルのＹ座標
	．ＣＸ	＄Ｄ０５９	カーソルのＸ座標
サブルーチン	ＥＲＡＳＥ１	＄Ｅ２Ａ０	Ａレジスタで与えられる番号の画面を消去する

解　説　ウィンドウ画面の画面番号をＡレジスタに代入して，ＥＲＡＳＥ１を呼び出して，ウィンドウ画面を消去します．ＥＲＡＳＥ１では，バッファポインタを更新しているので，それをカーソルの座標を示すワークエリアにコピーしています．この処理が終った後，終了キーコードのためにＡレジスタに２を代入して，ＧＥＴＣＲ１（＄ＥＣ４２）にジャンプします．ＧＥＴＣＲ１では，スタックに蓄えられているリターンアドレスを放棄して，ＧＥＴＣへ制御を移しています．

ＩＮＳＥＲＴ

＄ＥＡＥ０　～　＄ＥＡＥＢ

機　能	インサートモードの切換えを行います．		
レジスタ	ＣＣレジスタ以外使用されません．		
ワークエリア	．．ＩＮＳ	＄Ｄ０３３	インサートモードフラグ

解　説　このルーチンが呼ばれるたびに，．．ＩＮＳを反転させます．ＧＥＴＤＴ（＄Ｅ９８Ａ）では，．．ＩＮＳがクリアされている場合は普通の入力を行うだけですが，．．ＩＮＳがセットされていないときは，挿入動作も行っています．
なお，．．ＩＮＳを反転させて，＄００になるときは，インサートモードのインジケータ（ＬＥＤ）を消し，＄ＦＦになるときは，インジケータを点灯させてインサートモードであることを示します．
インジケータのＯＮ／ＯＦＦは，＄Ｄ４０Ｄのリード／ライトによって行います．

ＥＬＫＥＹ

＄ＥＡＥＣ　～　＄ＥＡ２０

機　能	現在のカーソルの位置から，次のフィールドの先頭の手前までの文字を消去します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	.ＢＰＹ	＄Ｄ０４５	バッファポインタのＹ座標
	.ＢＰＸ	＄Ｄ０４６	バッファポインタのＸ座標
	.ＢＰＡＤ	＄Ｄ０４７～＄Ｄ０４８	バッファポインタのキャラクタバッファアドレス
	.ＣＹ	＄Ｄ０５８	カーソルのＹ座標
	.ＣＸ	＄Ｄ０５９	カーソルのＸ座標
サブルーチン	ＢＵＦＰＴＲ	＄Ｅ１ＦＣ	バッファポインタのバッファアドレスを求める
	ＣＢＡＤＲ	＄Ｅ３Ｃ３	カーソルのバッファアドレスを求める
	ＣＤＳＰＣ	＄Ｅ５１８	カーソルのある位置に，キャラクタのフォントを表示する．
	ＣＳＲ	＄ＥＡ４４	カーソルを先に一つ進める

解　説　カーソルの位置のアトリビュートの変更フラグ（ビット６）をセットした後，次のフィールドの手前まで，Null コードを画面とキャラクタバッファの両方に出力していきます．この処理が終了したら，スタックに退避してあった元のカーソルの座標を取り出して，カーソルのワークエリアに設定します．なお，次のフィールドの先頭の座標が，バッファポインタの示す座標より大きい場合は，元のカーソルの座標で，バッファポインタの座標を設定します．

なお，アトリビュートバッファに格納されているアトリビュートが，現在画面に表示されているものと異るときは，このルーチンで Null を出力しているので，両者を同一のものにします．ただし，一つのフィールドに対してだけ行われます．

ＴＡＢＤＵＰ

ＧＥＴＤＵＰ	＄ＥＢ２１	～ ＄ＥＢ５５
ＴＡＢ	＄ＥＢ２Ｂ	
ＴＡＢＤＵＰ	＄ＥＢ２Ｃ	

機　能	ＴＡＢ／ＤＵＰ動作を行います．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ		
ワークエリア	..ＩＮＳ	＄Ｄ０３３	インサートモードフラグ
	..ＬＮＣ	＄Ｄ０３６	１行の桁数
	.ＭＤＲＧ	＄Ｄ０４Ｄ	ウィンドウ画面の開始行
	.ＣＹ	＄Ｄ０５８	カーソルのＹ座標
	.ＣＸ	＄Ｄ０５９	カーソルのＸ座標
	.ＣＢＡＤ	＄Ｄ０５Ａ～＄Ｄ０５Ｂ	カーソルのキャラクタバッファアドレス
	.ＣＡＡＤ	＄Ｄ０５Ｃ～＄Ｄ０５Ｄ	カーソルのアトリビュートバッファアドレス
サブルーチン	ＣＢＡＤＲ	＄Ｅ３Ｃ３	カーソルのバッファアドレスの設定
	ＴＡＢＣＨＫ	＄Ｅ７９Ｄ	与えられたＸ座標が，ＴＡＢ停止位置かどうかチェックする
	ＧＥＴＤＴ	＄Ｅ９８Ａ	ＧＥＴでの文字出力ルーチン

解　説　ＴＡＢ動作とＤＵＰ動作は，処理的には共通しているところが多くありますので，その中心となる部分は共通のルーチンＴＡＢＤＵＰで行っています．ＧＥＴＤＵＰでは，Ａレジスタに..ＬＮＣの値を代入し，符号を反転した後，ＴＡＢＤＵＰに制御を移します．ただし，現在のカーソルのＹ座標がウィンドウ画面の開始行と同じなら何もしないで終了します．

ＴＡＢでは，Ａレジスタをクリアして，ＴＡＢＤＵＰに制御を移します．

ＴＡＢＤＵＰでは，Ａレジスタがクリアされて与えられた場合はＴＡＢ動作として，ＳＰＡＣＥをＴＡＢ停止位置まで表示します．ＤＵＰのときは，Ａレジスタに１行の桁数を符号反転したものが代入されて与えられますので，それを用いて１行上の文字を表示させます．ＴＡＢ停止位置でなくても，保護フィールドや，フィールドの先頭にきたときも停止します．

なお，ＴＡＢＤＵＰでは，ＧＥＴＤＴを用いて文字を出力していますので，インサートモードのときＧＥＴＩＮＳに分岐しないように，インサートモードフラグ

をクリアしています（動作終了後は，元の値を戻しています）．

ＣＳＲＷ

＄ＥＢ５６　～　＄ＥＢ７４

機　能　カーソルを後方の英数文字列（グラフィック文字，カナ文字も含めます）の先頭へ移動させます．

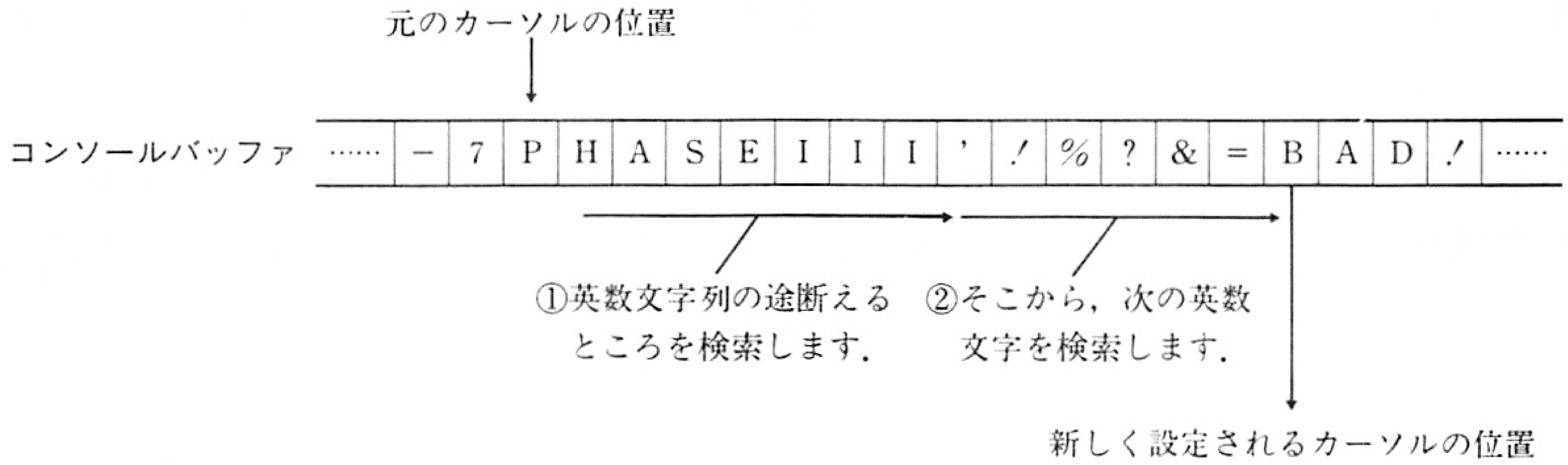
レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

ワークエリア
.ＭＤＲＧ　＄Ｄ０４Ｄ　ウィンドウ画面の開始行
ＧＴＢＴＭＣ　＄Ｄ０５２～＄Ｄ０５３　ウィンドウ画面のキャラクタバッファの終了アドレス
.ＣＹ　＄Ｄ０５８　カーソルのＹ座標
.ＣＸ　＄Ｄ０５９　カーソルのＸ座標
.ＣＢＡＤ　＄Ｄ０５Ａ～＄Ｄ０５Ｂ　カーソルのキャラクタバッファアドレス
.ＣＡＡＤ　＄Ｄ０５Ｃ～＄Ｄ０５Ｄ　カーソルのアトリビュートバッファアドレス

サブルーチン
ＧＥＴＥＮＣ　＄ＥＡ７９　キャラクタバッファアドレスから，ＸＹ座標を算出し，カーソルの座標として設定する．
ＣＳＲＷＳ　＄ＥＢ７５　英数文字列を検索する．

解　説　現在のカーソルの位置から後へ，英数文字列が途絶えるところを検索します．そして，その次の英数文字列が始まる場所を検索します．検索された場合は，そのキャラクタバッファアドレスをもとに，カーソルの座標を設定します．しかし，ウィンドウ画面の終端まで英数文字列の先頭が見つからない場合は，ラップしてウィンドウ画面の先頭にカーソルを持っていきます．

図３・４・19　ＣＳＲＷの動作例



ＣＳＲＷＳ

＄ＥＢ７５　～　＄ＥＢ８Ａ

機　　能	英数文字を検索していきます．	
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ	
入力情報	Ｘレジスタ	検索をする座標のキャラクタバッファアドレス
	Ｙレジスタ	検索をする座標のアトリビュートバッファアドレス
出力情報	Ｃフラグ	英数文字列（カナ文字，グラフィック文字も含む）のときセットされます
	Ｘレジスタ	次の検索をする座標のキャラクタバッファアドレス
	Ｙレジスタ	次の検索をする座標のアトリビュートバッファアドレス
ワークエリア	ＧＴＦＬＤ	＄Ｄ０Ａ５　フィールドの先頭に来たときセット（≠０）されます．
サブルーチン	ＣＳＷＣＫ	＄ＥＢ８Ｃ　英数文字かどうかをチェック

解　　説　このルーチンは，ＣＳＲＷ（＄ＥＢ５６）のための局所的なサブルーチンです．ＣＳＲＷに繰り返し呼ばれることによって，英数文字列をスキップしたり，逆に英数文字列の先頭を見つける役割を受け持っています．

入力情報で与えられた座標の文字をチェックし，Ｘレジスタ，Ｙレジスタを＋１します．また，フィールドの先頭にきたときはＧＴＦＬＤをセットします．ＧＴＦＬＤがセットされた状態でこのルーチンをＣＳＲＷが呼びだすと，リターンアドレスを放棄して，ＣＳＲＷ９０（＄ＥＢ６７：ＣＳＲＷの途中のエントリ）に強制的にジャンプします．

ＣＳＷＣＫ

＄ＥＢ８Ｃ　～　＄ＥＢＡ３

機　　能	英数文字（グラフィック文字，カナ文字）かどうかチェックします．	
レジスタ	Ａ，Ｂ，Ｘ，Ｕ	
入力情報	Ｘレジスタ	検索をする座標のキャラクタバッファアドレス
出力情報	Ｃフラグ	英数文字，グラフィック文字，カナ文字の場合にセットされます．

解　説　文字をチェックするために，＄ＥＢＤ２～＄ＥＢＤＡ番地の領域(ラベル名ＣＳＷＫＴ）に，検索文字の開始コードのテーブルを持っています．キャラクタコードでは，数字と英字とグラフィック・カナ文字とは，連続しているコードになっていないので，各々の文字の開始コードと終了コードの対を計４個持っています．調査する文字のコードが，これらの範囲内に入っていればＣフラグをセットしますし，範囲外ならＣフラグをクリアします．

ＣＳＬＷ

＄ＥＢＡ４　～　＄ＥＢＢ８

機　能　カーソルを前方の英数文字列の先頭へ移動します．

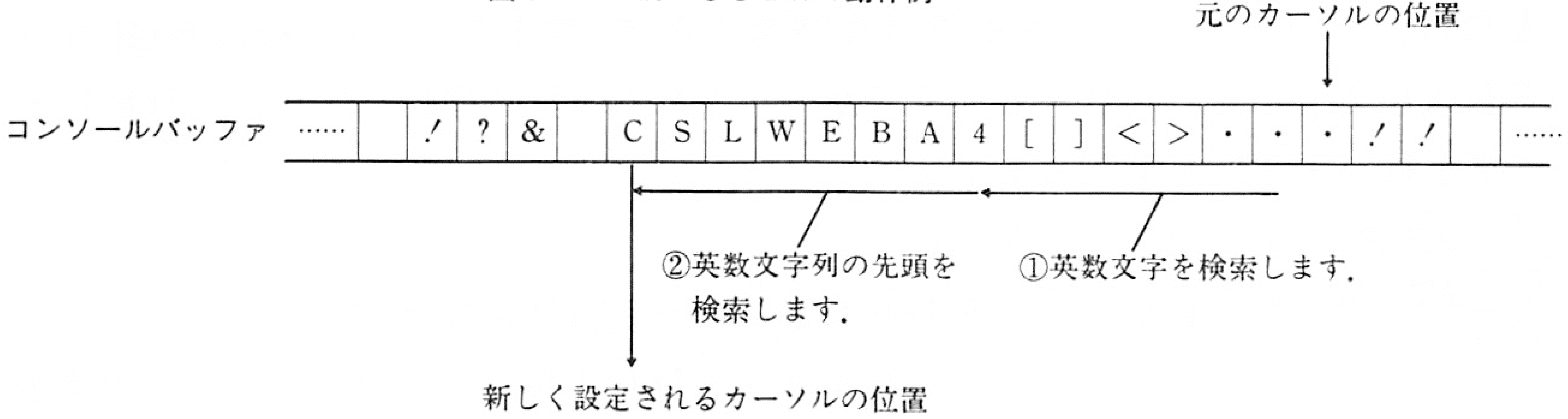
レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

ワークエリア　ＧＴＦＬＤ　＄Ｄ０Ａ５　文字列検索の際，フィールドの先頭にきたときにセットされます

サブルーチン　ＣＢＡＤＲ　＄Ｅ３Ｃ３　カーソルのバッファアドレスを求める
ＣＳＬ　＄ＥＡ６０　カーソルを１つ前に戻す
ＧＥＴＥＮＡ　＄ＥＡ７Ｆ　アトリビュートバッファアドレスからカーソルの座標を設定する
ＣＳＬＷＳ　＄ＥＢＢ９　英数文字を検索する

解　説　現在のカーソルの位置の手前から前方に，英数文字を検索していき，英数文字が発見されれば，英数文字の先頭を検索します．検索された場合は，その場所をカーソルの位置として再設定します．ただし，フィールドの区切りにきた場合は，フィールドの先頭をカーソルの位置として設定します．

図３・４・20　ＣＳＬＷの動作例



ＣＳＬＷＳ

＄ＥＢＢ９　～　＄ＥＢＤ１

機　能	英数文字列を検索していきます．	
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ	
入力情報	Ｘレジスタ	検索する座標のキャラクタバッファアドレス
	Ｙレジスタ	検索する座標のアトリビュートバッファアドレス
出力情報	Ｃフラグ	英数文字の場合セットされる
	Ｘレジスタ	検索した座標の次のキャラクタバッファアドレス
	Ｙレジスタ	検索した座標の次のアトリビュートバッファアドレス
ワークエリア	ＧＴＦＬＤ	＄Ｄ０Ａ５　フィールドの先頭通過フラグ
サブルーチン	ＣＳＷＣＫ	＄ＥＢ８Ｃ　英数文字かどうかチェックする

解　説　このルーチンはＣＳＬＷ専用のルーチンで，英数文字列の先頭や終端を検索するのに用いられます．与えられた座標が英数文字かどうかをＣＳＷＣＫを用いて調べた後，Ｘ，Ｙレジスタを－１します．フィールドの先頭にきた場合，次にこのルーチンを呼ぶとＣＳＬＷ９０（＄ＥＡ７Ｆ）へ強制ジャンプして，その位置のバッファアドレスから，カーソルのＸＹ座標を設定します．

ＧＥＴＤＥＬ

ＧＥＴＢＳ　　＄ＥＢＤＣ　～　＄ＥＣ３Ｄ
ＧＥＴＤＥＬ　＄ＥＢＥ８

機　能　ＤＥＬオーダ（カーソル位置の文字を削除して，Null コードの直前または，フィールドの最後までの文字を１文字シフトして，その後にNull コードを入れます）またはＢＳオーダ（カーソルを１つ左に移動してから，ＤＥＬオーダを行います）を実行します．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ	
ワークエリア	ＫＬＯＣＫ	＄Ｄ００２　キー入力禁止フラグ
	．ＢＡＡＤ	＄Ｄ０４９～＄Ｄ０４Ａ　バッファポインタのアトリビュートバッファアドレス
	．ＣＹ	＄Ｄ０５８　カーソルのＹ座標
	．ＣＸ	＄Ｄ０５９　カーソルのＸ座標

.ＣＢＡＤ　＄Ｄ０５Ａ～＄Ｄ０５Ｂ　カーソルのキャラクタバッファアドレス

.ＣＡＡＤ　＄Ｄ０５Ｃ～＄Ｄ０５Ｄ　カーソルのアトリビュートバッファアドレス

サブルーチン	ＣＢＡＤＲ	＄Ｅ３Ｃ３	カーソルのバッファアドレスの設定
	ＣＤＳＰＣ	＄Ｅ５１８	カーソルの位置にフォントを表示
	ＣＴＯＢ	＄Ｅ７８７	カーソルの位置をバッファポインタへ
	ＣＳＲ	＄ＥＡ４４	カーソルを１つ先に進める
	ＣＳＬ	＄ＥＡ６０	カーソルを１つ前に戻す

解　説　ＧＥＴＢＳでは，ＣＳＬを用いてカーソルを１つ前に戻しておいてＣＢＡＤＲでそのバッファアドレスを設定してから，ＧＥＴＤＥＬへ制御を移します．ただし，カーソルがフィールドの先頭にきている場合は，何も行いません．

ＧＥＴＤＥＬでは，カーソルの位置以降の文字を１文字左へシフトさせています．ただし，保護フィールドにカーソルがあった場合は何も行いません．また，シフトされる文字列の途中にバッファポインタがあった場合，シフトされた文字列の後にバッファポインタを設定しています．なお，シフトしている最中は，キーをロックしています．これは，ＤＥＬに時間がかかるので，その間は，キー入力バッファにキーコードを蓄えないようにするためです．処理できる範囲でキー入力を扱うようにするという意味から行われています．

ＧＥＴＣＲ

ＧＥＴＣＡＮ	＄ＥＣ３Ｅ	～	＄ＥＣＡＢ
ＧＥＴＣＲ	＄ＥＣ４１		
ＧＥＴＣＲ１	＄ＥＣ４２		

機　能　キー入力（スクリーンエディット）を終了させます．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

ワークエリア　.ＥＫＥＹ　＄Ｄ０３４　終了キー番号

解　説　ＧＥＴＣＡＮエントリでは，ＥＫＥＹに１を，ＧＥＴＣＲエントリでは．ＥＫＥＹに０を，ＧＥＴＣＲ１エントリでは．ＥＫＥＹにＡレジスタの値を代入しています．ＧＥＴＣＲ１は，ＧＥＴＣＬＲ（＄ＥＡＤ２）用のエントリ

でＡレジスタの値を２にしてエントリしてきます．
その後はリターンアドレスを放棄して，ＧＥＴＣ（＄ＥＣ４６）へ制御を移します．ＧＥＴＣについては３－４－３節を参照して下さい．

３－４－５　ＰＵＴ，ＧＥＴ関連コマンドルーチン

(1) Get Buffer Address (GETBAD) $0A ($ECEC)

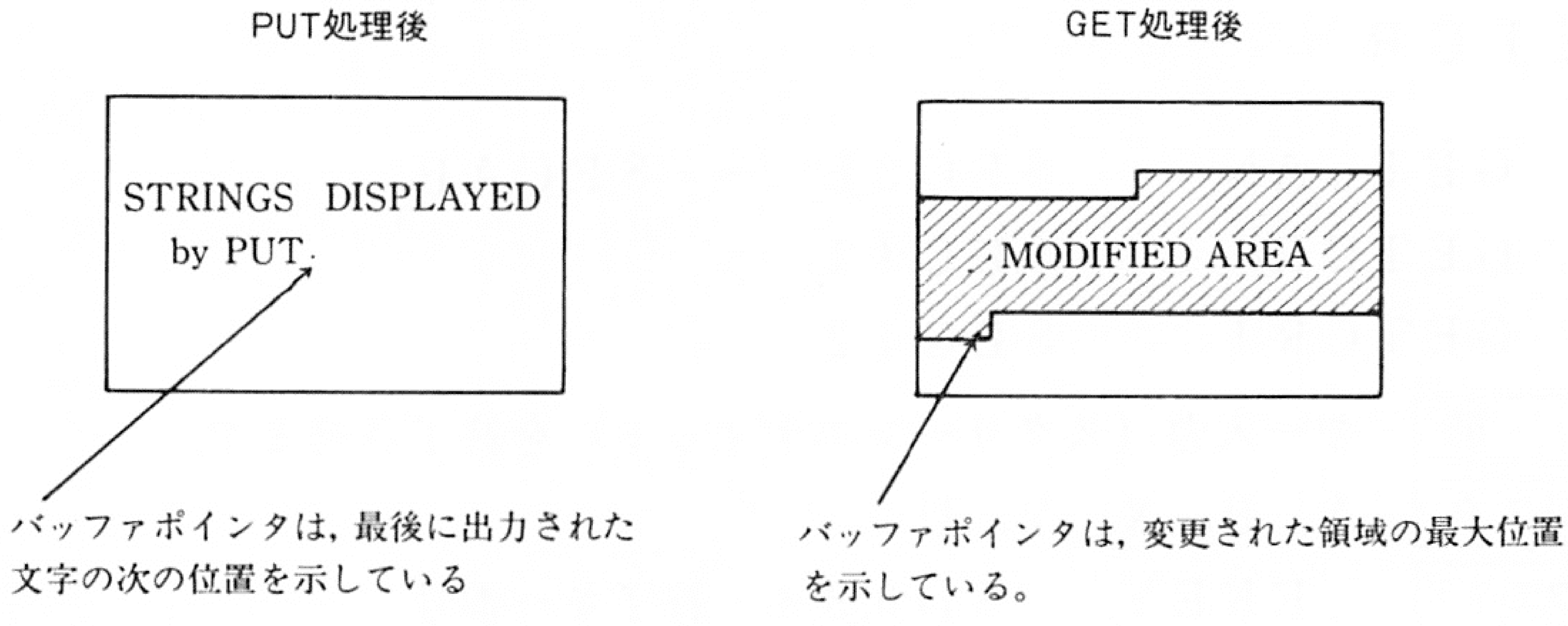
現在のバッファポインタが示しているバッファアドレスを読み取ります．

共有ＲＡＭ上の復帰情報

アドレス	記　号	内　　　容	値
Ｄ３８３	Ｘ	バッファポインタＸ座標	０～79(キャラクタ座標)
Ｄ３８４	Ｙ	バッファポインタＹ座標	０～24(キャラクタ座標)

解　　説　バッファポインタは，ＰＵＴコマンドが終ったときは，出力文字の最終位置が入っていますし，ＧＥＴコマンドが終ったときは，変更された部分のうちでアドレスが最大となる位置が入っています．このコマンドは，ＰＵＴやＧＥＴで行われた文字操作処理の最終座標を知るためにあります．そして，その最終座標というのは，次の処理を始める先頭座標でもあります．

図３・４・21　PUT, GET処理後のバッファポインタ



動作は次のようになります．

１）バッファポインタのＸ座標（．ＢＰＸ：＄Ｄ０４６）をＸ（＄Ｄ３８３）に，Ｙ座標（．ＢＰＹ：＄Ｄ０４５）をＹ（＄Ｄ３８４）に格納します．

(2) TABSET　$0B　($EDBC)

TAB（およびDUP）キーを押したときの，停止位置を設定します

共有RAM上のパラメータ

アドレス	記　号	内　容	値
D383 〜 D38C	TAB	TAB設定テーブル	80ビットのビットイメージデータ

主なサブルーチン

$E44A　MOVE　ブロック転送ルーチン

解　説　10バイト（80ビット）のデータを画面の1行分のイメージとして，ビットが1になっている位置をTAB（DUP）キーの停止位置とします．先頭バイトのMSBを1行の先頭（左端），最終バイトのLSBを1行の末尾（右端）とします．

このルーチンは，共有RAM上にあるTAB（$D383〜$D38C）を，MOVEを用いて，ワークエリア .TABST（$D024〜$D02D）へ転送して終了します．

ワークエリア .TABSTは，TAB設定テーブルとして，TABのオーダルーチンにおいて停止位置かどうかを判別するのに用いられています．

図3・4・22　TAB設定テーブル（共有RAM上）のビットパターンと停止位置

X座標	0 〜 7	8 〜 15	16 〜 23	24 〜 31	32 〜 39	40 〜 47	48 〜 55	56 〜 63	64 〜 71	72 〜 79
テーブル	D383	D384	D385	D386	D387	D388	D389	D38A	D38B	D38C
	(D024)	(D025)	(D026)	(D027)	(D028)	(D029)	(D02A)	(D02B)	(D02C)	(D02D)

テーブルのビットが1になっている座標で，TABが停止します．

(3) Console Control（CNSCTL）$0C（$EDC8）

種々のコンソール機能を選択します．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
Ｄ３８３	ＣＦ	制御フラグ	ビット０：カーソル表示　１＝する　０＝しない ビット１：オーダ動作　１＝する　０＝しない ビット２：ＴＡＢ動作　１＝する　０＝しない ビット３：ページウェイト　１＝する　０＝しない ビット４：プットウェイト　１＝する　０＝しない ビット５：オートＬＦ　１＝する　０＝しない

主なサブルーチン

＄ＥＤＤ３　ＣＵＲＯＮ　　カーソルの点滅を開始させる

＄ＥＤＥ３　ＣＵＲＯＦＦ　カーソルを消去し，点滅を禁止する

＄ＥＤＥ８　ＣＵＲＳＥＴ　カーソルの位置を設定する

解　説　動作は，ＣＦ（＄Ｄ３８３）の内容をワークエリア中のコンソール制御フラグ．ＣＳＣＴＬ（＄Ｄ０２１）に移すだけです．この．ＣＳＣＴＬは，ＰＵＴ，ＧＥＴの各ルーチンで参照されています．ただ，カーソルの点滅・消去はすぐに行う必要があるので，表示させるときは，ＣＵＲＳＥＴでカーソルの位置を設定してから，ＣＵＲＯＮによって点滅を開始させます．また，消去するときは，ＣＵＲＯＦＦを用いています．

このルーチンには，ＣＮＳＩＮＴ（＄ＥＤＣＢ）というエントリがあり，任意にコンソール制御フラグを設定することができます．このエントリで呼び出すときは，Ａレジスタに設定する内容を入れておく必要があります．

コンソール制御フラグは，ビット０～５までが有効なのですが，その他にビット７も参照されています．これは，ＰＵＴにおいて，画面最右端に文字を表示した後の自動改行の禁止フラグで，ビット７がセットされていると，最右端に文字を表示した後は，同じ行の先頭から残りの文字列を表示していくことになります．

ビット７は，ＰＵＴＩＮＣ（＄Ｅ７ＢＡ）で参照されていますので，そちらも参考にして下さい．

３－４－６　Character Line ＄２０（＄ＥＥ７２）

キャラクタ座標（Ｘ１，Ｙ１）から（Ｘ２，Ｙ２）まで，文字を使って直線または，四角を描きます．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　　容	値
Ｄ３８３	ＣＬ	カラーコード	０～７
Ｄ３８４	ＣＨＲ	文字コード	＄２０～＄ＦＥ
Ｄ３８５	－		０
Ｄ３８６	Ｘ１	始点のＸ座標	０～７９(キャラクタ座標)
Ｄ３８７	－		０
Ｄ３８８	Ｙ１	始点のＹ座標	０～２４(キャラクタ座標)
Ｄ３８９	－		０
Ｄ３８Ａ	Ｘ２	終点のＸ座標	０～７９(キャラクタ座標)
Ｄ３８Ｂ	－		０
Ｄ３８Ｃ	Ｙ２	終点のＹ座標	０～２４(キャラクタ座標)
Ｄ３８Ｄ	Ｂ	ボックスフラグ	０＝直線 １＝四角形の枠 ２＝四角形塗りつぶし

主なサブルーチン

＄ＥＥ９３	ＬＩＮＥＤＯ	ＬＩＮＥメインルーチン
＄Ｅ３Ｄ５	ＣＡＸＣＲ	キャラクタ座標の範囲をチェック
＄Ｅ２０Ｃ	ＢＡＤＲ	キャラクタ座標からキャラクタバッファのアドレスおよびアトリビュートバッファのアドレスを求める
＄Ｅ２２５	ＢＶＡＤ	キャラクタ座標からＶＲＡＭアドレスを求める
＄Ｅ５１Ｄ	ＣＤＳＰ	キャラクタフォントの出力

※ＬＩＮＥＤＯは，ＬＩＮＥの下位ルーチンで＄Ｄ０Ａ０の値によってキャラクタラインを実行します

補助ワークエリア

＄Ｄ０Ａ０	ＬＮＣＨＲ	０…ライン それ以外…キャラクタライン

※他のワークエリアについてはＬＩＮＥ，ＬＩＮＥＳを参照して下さい．

解　説　このルーチンはＬＩＮＥのメインルーチンを使用しています．この使いわけをするのがＬＮＣＨＲ（＄Ｄ０Ａ０）というフラグです．このルーチンではＸ座標に平行な線とＹ座標に平行な線を描くのを区別していません．つまり，ＬＩＮＥＳの３）のルーチンのみを使っているのです．

３－４－７　ＰＵＴ，ＧＥＴ汎用サブルーチン

ＣＤＳＰ

ＣＤＳＰＣ	＄Ｅ５１８　～　＄Ｅ５８Ｅ
ＣＤＳＰＢ	＄Ｅ５１Ｂ
ＣＤＳＰ	＄Ｅ５１Ｄ

機　能	キャラクタを画面に出力します．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	Ａレジスタ	キャラクタコード	
	Ｂレジスタ	アトリビュートコード	
	Ｘレジスタ	ＶＲＡＭ上のアドレス（ＣＤＳＰのみ）	
出力情報	Ｘレジスタ	キャラクタのフォントのアドレス	
	Ｕレジスタ	ＶＲＡＭ上のアドレス	
ワークエリア	ＣＳＬＣＤＳ	＄Ｄ０２３	単色表示フラグ
	..ＢＣＬ	＄Ｄ０２Ｆ	現在の背景色カラーコード
	..ＣＷＤ	＄Ｄ０３１	キャラクタ幅
	.ＢＶＡＤ	＄Ｄ０４Ｂ～＄Ｄ０４Ｃ	（ＣＤＳＰＢのみ）
	.ＣＶＡＤ	＄Ｄ０５Ｅ～＄Ｄ０５Ｆ	（ＣＤＳＰＣのみ）
	ＨＫＦＬＧ	＄Ｄ０９５～＄Ｄ０９６	
	ＨＫＢＬＵＥ	＄Ｄ０９７～＄Ｄ０９９	
	ＨＫＲＥＤ	＄Ｄ０９Ａ～＄Ｄ０９Ｃ	
	ＨＫＧＲＥＥ	＄Ｄ０９Ｄ～＄Ｄ０９Ｆ	
サブルーチン	ＨＫＢＬＵＥ	＄Ｄ０９７	青の画面に対する表示ルーチン
	ＨＫＲＥＤ	＄Ｄ０９Ａ	赤の画面に対する表示ルーチン
	ＨＫＧＲＥＥ	＄Ｄ０９Ｄ	緑の画面に対する表示ルーチン
	ＣＦＡＤＲ	＄Ｅ３Ｂ７	フォントのアドレスを求める
	ＣＤＳＩＮＴ	＄Ｅ４Ｅ６	ジャンプテーブルの初期化（設定）
	ＢＩＴＥＸＴ	＄Ｅ５０Ｂ	フォントの拡張用のルーチン

CDSRVL～　＄Ｅ５８Ｆ　フォントの出力サブルーチン群

解　説　エントリによって次の違いがでてきます．

１）ＣＤＳＰＣ　　カーソルの（．ＣＶＡＤが示す）位置にキャラクタを出力します．

２）ＣＤＳＰＢ　　バッファポインタの（．ＢＶＡＤが示す）位置にキャラクタを出力します．

３）ＣＤＳＰ　　ＶＲＡＭ上のアドレスをＸレジスタが示す位置にキャラクタを出力します．

キャラクタが出力されたＶＲＡＭ上のアドレスは，Ｕレジスタに保管され，ルーチンの終了時に出力情報としてＵレジスタに設定されます．

動作は次のようになります．

１）キャラクタのフォントのアドレスを求めます（ＣＦＡＤＲ）．

２）ワークエリアＣＳＬＣＤＳ（＄Ｄ０２３）によって，カラーモードかどうかを判別し，各々の場合によって次のように色を決めていきます．

　ｉ）カラーモードの場合，反転表示のときは文字色と背景色（ワークエリア..ＢＣＬ）を取り換えます．

　ⅱ）単色表示モードの場合，普通の表示のときは，文字色に緑(＝４)，背景色に黒（＝０）を代入します．反転表示のときは，文字色に黒，背景色に緑を代入します．

３）設定された文字色と背景色が今まで表示されたもの（ワークエリアＨＫＦＬＧに保管されています）と異るときは，ＣＤＳＩＮＴルーチンを用いて，各画面に対するフォント出力ルーチンへのジャンプテーブル（ＨＫＢＬＵＥ，ＨＫＲＥＤ，ＨＫＧＲＥＥ）を新たに設定します．

４）ワークエリア..ＣＷＤによって，キャラクタのサイズが大きい（１行４０桁のとき）か小さい（１行８０桁のとき）か判別し，大きいときはＢＩＴＥＸＴルーチンを用いて，キャラクタのフォントを横に２倍に拡大したものをワークエリアＥＸＦＯＮＴ（＄Ｄ０８３～＄Ｄ０９２）に設定します．

５）各画面に対するフォント出力ルーチンを呼び出して，各画面にキャラクタのフォントを出力していきます．赤，緑の画面に書くときは，ＶＲＡＭ上のアドレスに＄４０００，＄８０００のオフセットを加えています．なお，単色表示モードは，緑の画面にしか出力を行っていません．

各画面に対する出力ルーチン群は，＄E５D０番地以降にあるのですが，文字色，背景色および反転表示の選択および文字の大小の各状態に応じて，各画面への出力ルーチンが変ってきます．このためRAM（ワークエリア）上に各ルーチンへジャンプするためのジャンプ命令とジャンプベクトルを組み合わせたもの(フック）を設けておいて，そこを呼びだすことによって間接的に各ルーチンを呼び出しています．このフック（ジャンプテーブル）がHKBLUE，HKRED，HKGREEで，各々青，赤，緑の画面に対する出力ルーチンのエントリへジャンプするように設定されています．

例えば，文字色が緑，背景色が紫（＝青＋赤）の反転表示で，文字のサイズが小さい（１行が８０桁）ときは，次のように各フックを設定します．

１）青の画面出力ルーチン用のフック（HKBLUE）

キャラクタのフォントをそのまま出力するルーチンを呼びだすようにします．

２）赤の画面出力ルーチン用のフック（HKRED）

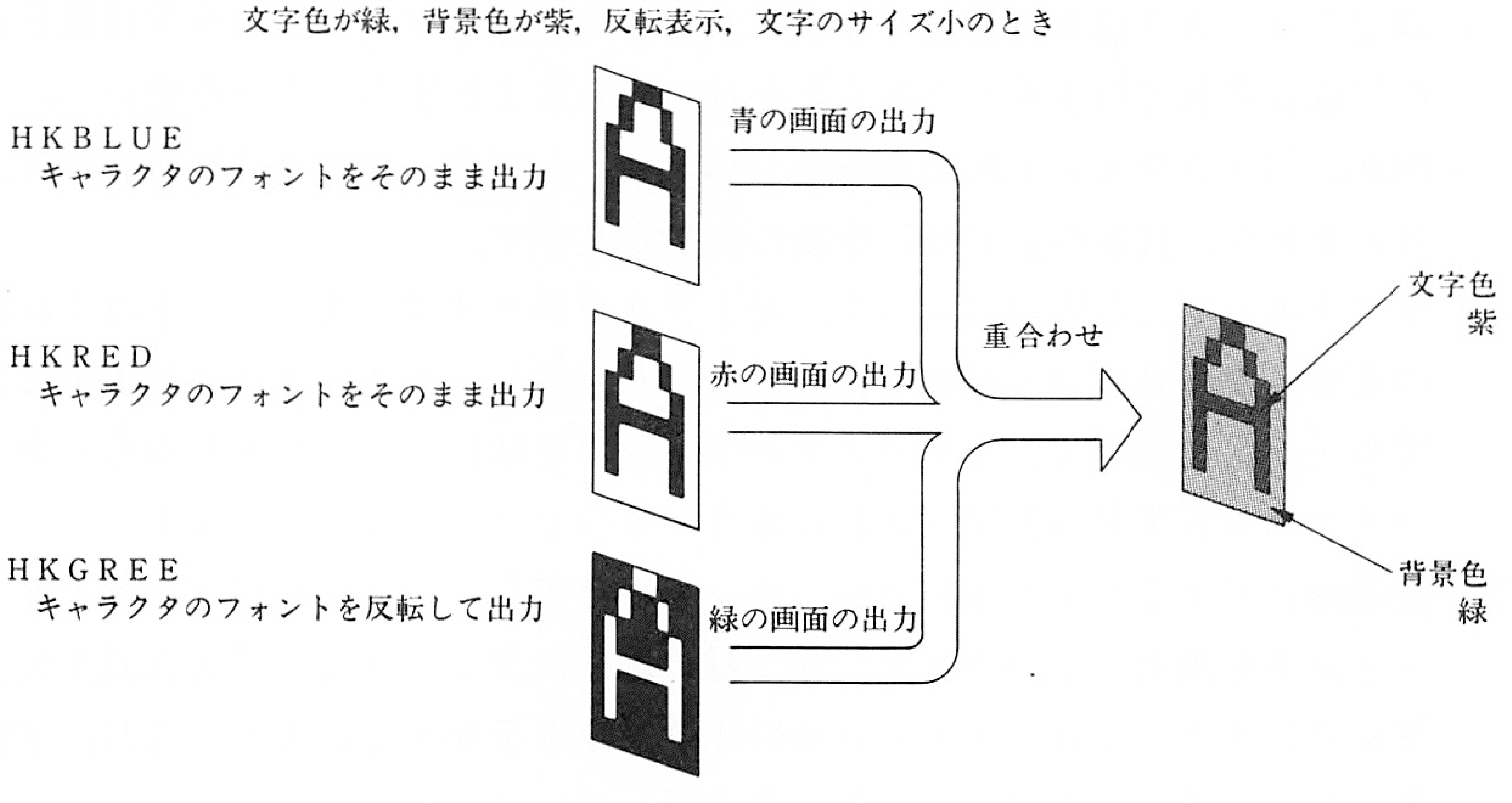
キャラクタのフォントをそのまま出力するルーチンを呼びだすようにします．

３）緑の画面出力ルーチン用のフック（HKGREE）

キャラクタのフォントを反転させて出力するルーチンを呼びだすようにします．

このように設定すると，文字色が紫，背景色が緑のフォントを出力することができます（図３・４・23）．詳細はCDSINT，CDSRVLを参照して下さい．

図３・４・23　キャラクタの出力例

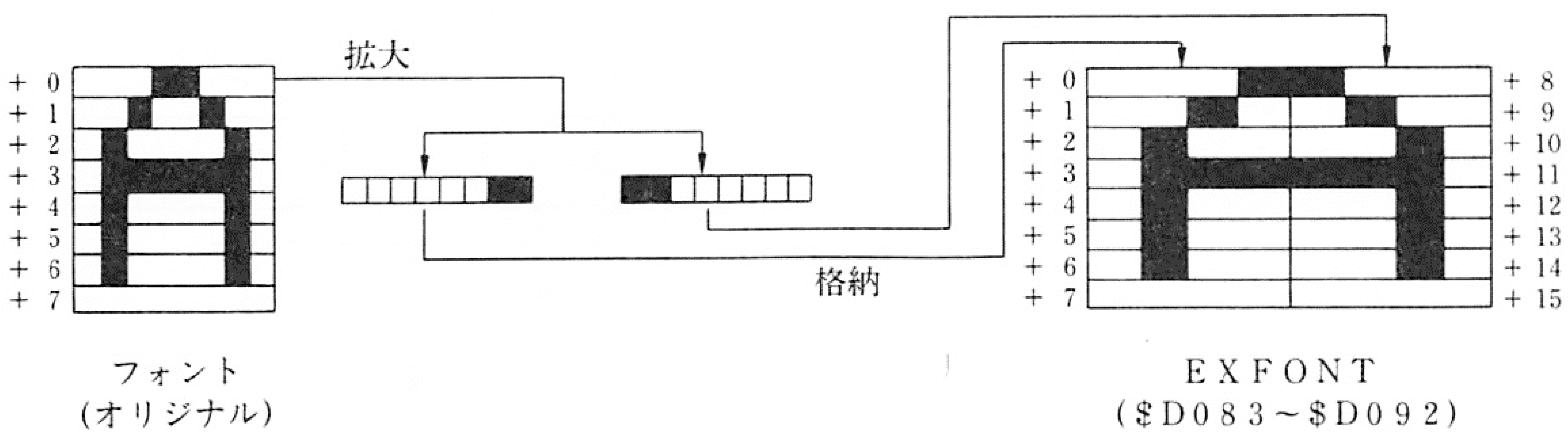


キャラクタのサイズが大きいとき（1行40桁）は，BITEXTを用いてキャラクタの幅を横方向に2倍に拡大しています．拡張されたフォントはEXFONT（$D083～$D092）に格納されますが，EXFONTの前半の8バイトはキャラクタの左半分，後半の8バイトは右半分のフォントが設定されることになります（図3・4・24）．具体的な動作は次のようになります．

1）キャラクタのフォントを1バイト拾ってきて，上位4ビットを8ビットに拡張したものをEXFONTに格納します．また下位4ビットを8ビットに拡張したものを先程格納したワークエリアの8バイト後の番地に格納します．

2）以上の動作を8回繰り返していきます．

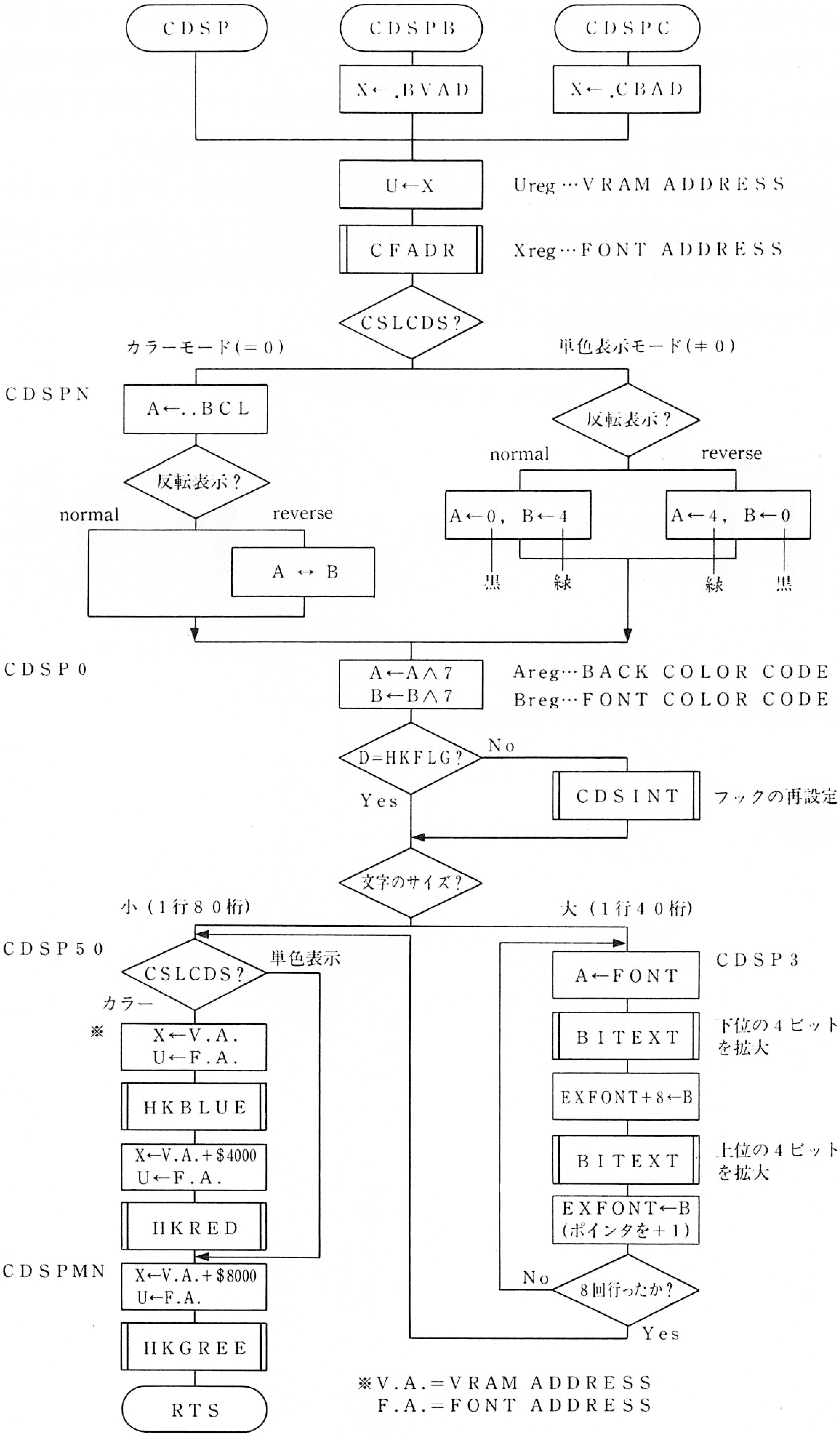
図3・4・24　キャラクタの拡大



次の例は水色で大文字のMを表示します．

```
LDA   #'M
LDB   #5
LDX   #$2000
JSR   CDSP
```

図3・4・25　CDSPの処理



ＣＤＳＩＮＴ

＄Ｅ４Ｅ６　～　＄Ｅ５０Ａ

機　能　ＣＤＳＰ（＄Ｅ５１Ｄ）の画面出力サブルーチンを呼び出すためのフックを再設定します．

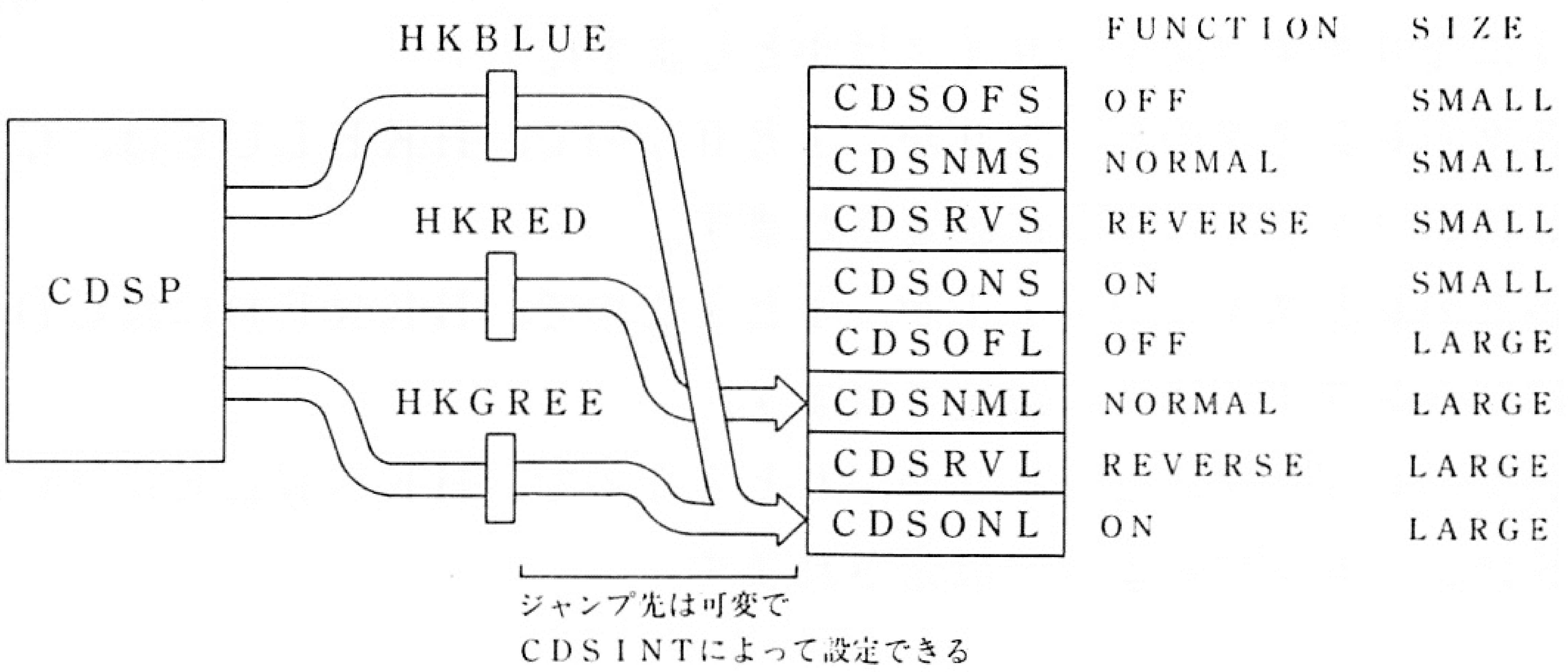
レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ　（Ａ，Ｂ，Ｘは保存）

入力情報　Ａレジスタ　背景色（０～７）
Ｂレジスタ　文字色（０～７）

ワークエリア　..ＣＷＤ　＄Ｄ０３１　キャラクタの幅
ＨＫＦＬＧ　＄Ｄ０９５～＄Ｄ０９６
ＨＫＢＬＵＥ　＄Ｄ０９７～＄Ｄ０９９
ＨＫＲＥＤ　＄Ｄ０９Ａ～＄Ｄ０９Ｃ
ＨＫＧＲＥＥ　＄Ｄ０９Ｄ～＄Ｄ０９Ｆ
ＣＤＪＭＰ　＄Ｅ５Ｄ０～＄Ｅ５ＤＦ　ジャンプベクトルデータ

解　説　文字色，背景色および文字のサイズによって，各フックのジャンプ先を変えていきます．ジャンプベクトルのデータは，ＣＤＪＭＰ（＄Ｅ５Ｄ０～＄Ｅ５ＤＦ）に格納されています．なお，この時点での文字色と背景色は実際のものであり，反転表示は考慮されていません．

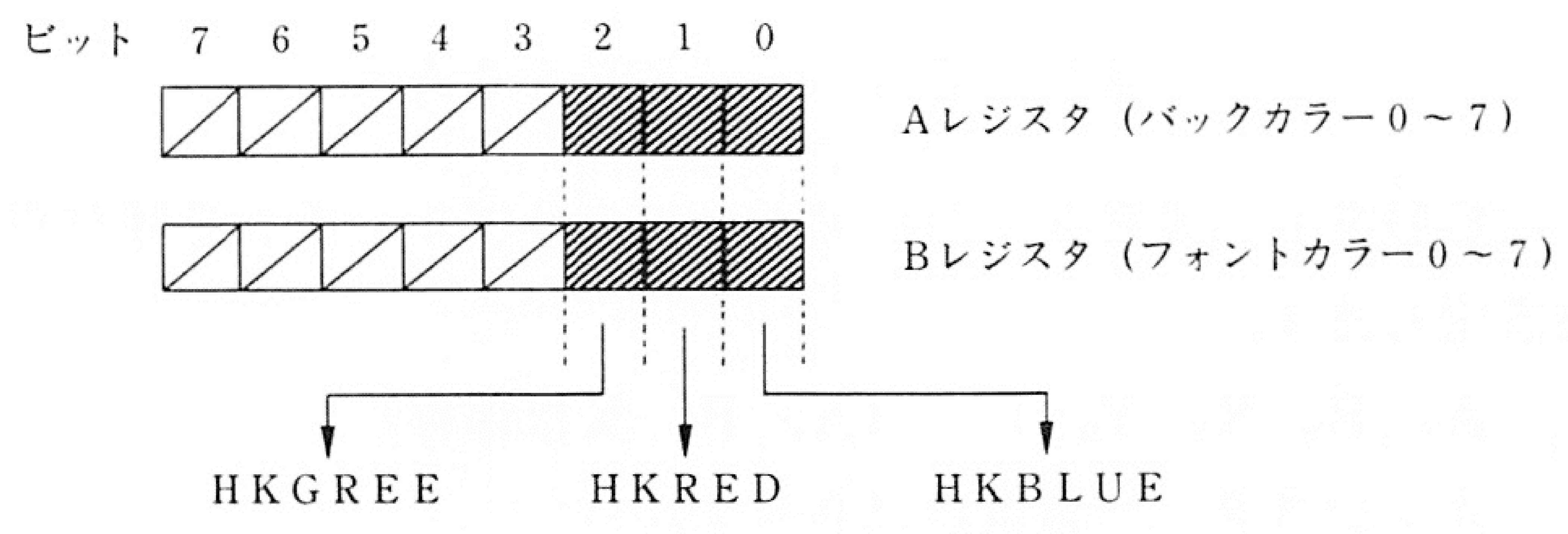
図３・４・26　ＣＤＳＰのフックを通してのサブルーチンのコール



※この例では，文字色は白色，背景色は水色，文字のサイズは大になります

各フックを設定する際に参照されるのは，Ａ，Ｂレジスタの下位３ビットで，ＨＫＢＬＵＥがビット０，ＨＫＲＥＤがビット１，ＨＫＧＲＥＥがビット２によって決定されます．

図3・4・27　A，Bレジスタの各ビットによるフックの設定



A，Bレジスタの各ビットのON（＝1），OFF（＝0）によって図3・4・28のようにジャンプベクトルが設定されます．なお，ジャンプベクトルが設定されるのは，各フックの2バイト目と3バイト目で，1バイト目にはジャンプ命令（JMP：＄7E）が設定されます．

図3・4・28　A，Bレジスタの各ビットによるジャンプベクトルの選択

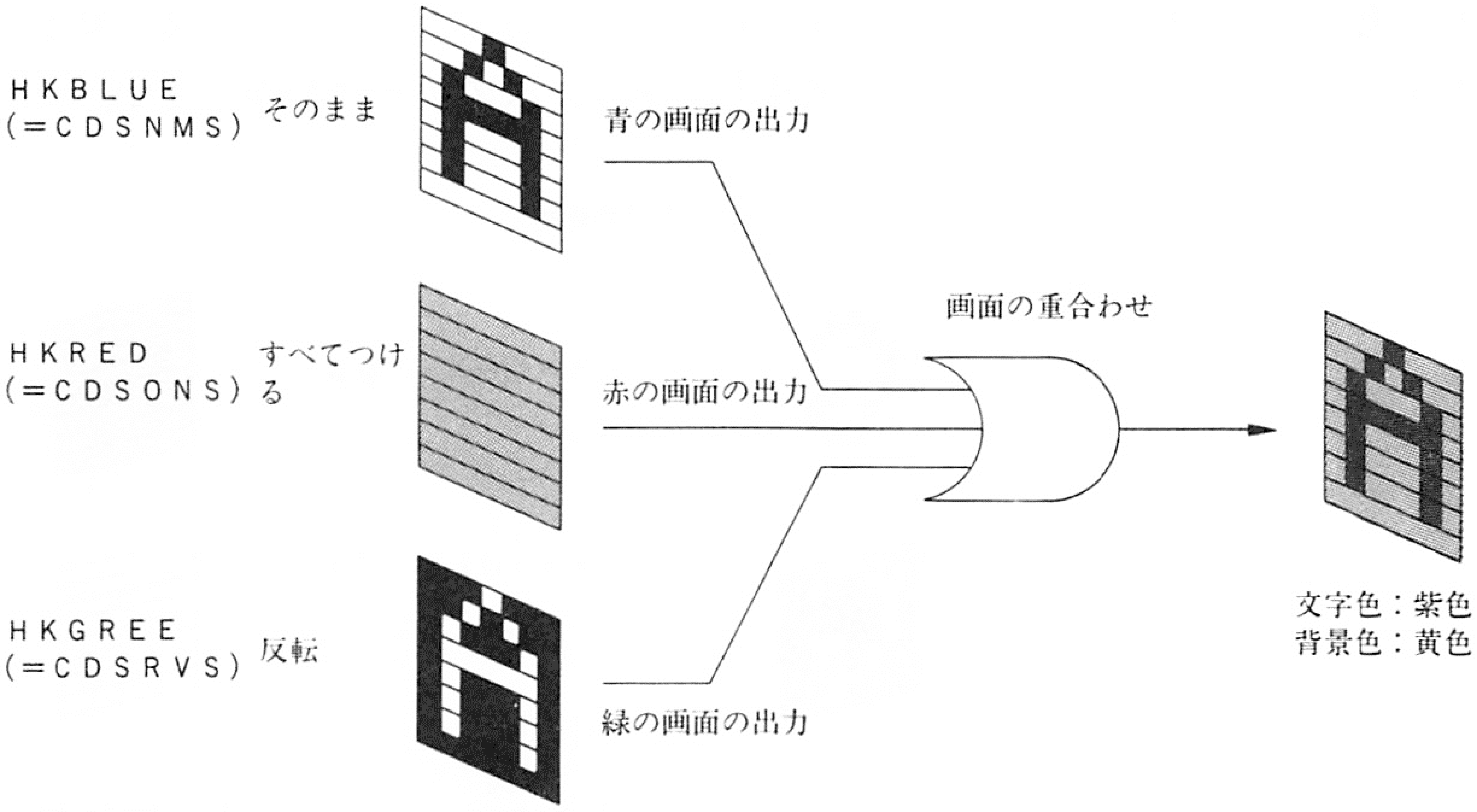
Aレジスタのビット	Bレジスタのビット	ジャンプベクトル 文字のサイズ小(S)	ジャンプベクトル 文字のサイズ(L)	内容
0	0	E5B2	E5C6	CDSOF　すべてのドットを消す
0	1	E59A	E590	CDSNM　フォントを表示する
1	0	E599	E58F	CDSRV　フォントを反転させて表示する
1	1	E5B1	E5C5	CDSON　すべてのドットをつける

例えば，Aレジスタに3（紫），Bレジスタに6（黄）が設定された場合は，次のようになります（文字のサイズは小とします）．

1）各々のレジスタのビット0が，1と0なので，HKBLUEは，CDSNMSにジャンプするように設定されます．
2）各々のレジスタのビット1が，1と1なので，HKREDにはCDSONSにジャンプするように設定されます．
3）各々のレジスタのビット2が，0と1なので，HKGREEにはCDSRVSにジャンプするように設定されます．

このように，青の画面では，フォントのままにドットをつけ，赤の画面ではすべてのドットをつけ，緑の画面では，フォントのところだけドットを消すことによって，文字色が黄，背景色が紫のキャラクタを出力することになります．

図3・4・29　各画面の重ね合わせ



※大文字のAを出力しています

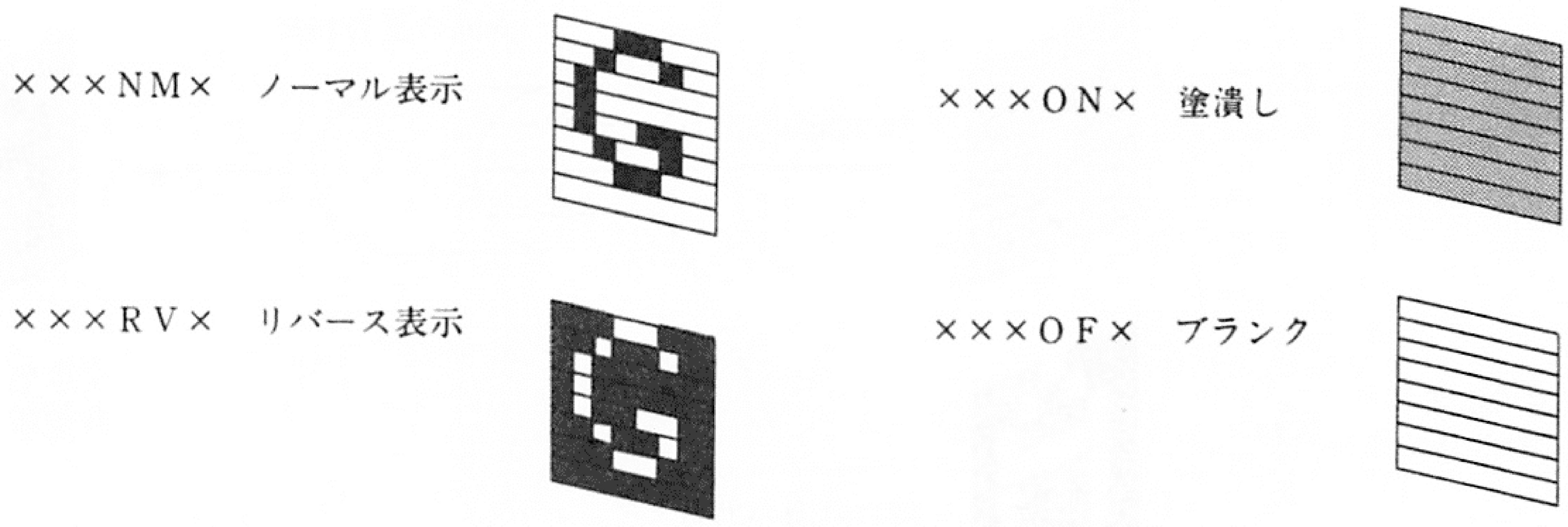
CDSRVL

CDSRVL	$E58F	～	$E5C4
CDSNML	$E590		
CDSRVS	$E599		
CDSNMS	$E59A		
CDSONS	$E5B1		
CDSOFS	$E5B2		
CDSONL	$E5C5	～	$E5CF
CDSOFL	$E5C6		

機　能	各画面に対するキャラクタのフォントの出力ルーチン群です.	
レジスタ	A，B，X，U	
入力情報	Dレジスタ	#$0008
	Xレジスタ	VRAMのアドレス（赤，緑の画面含む）
	Uレジスタ	フォントのアドレス
ワークエリア	..CLEX	$D030
	..CHT	$D032

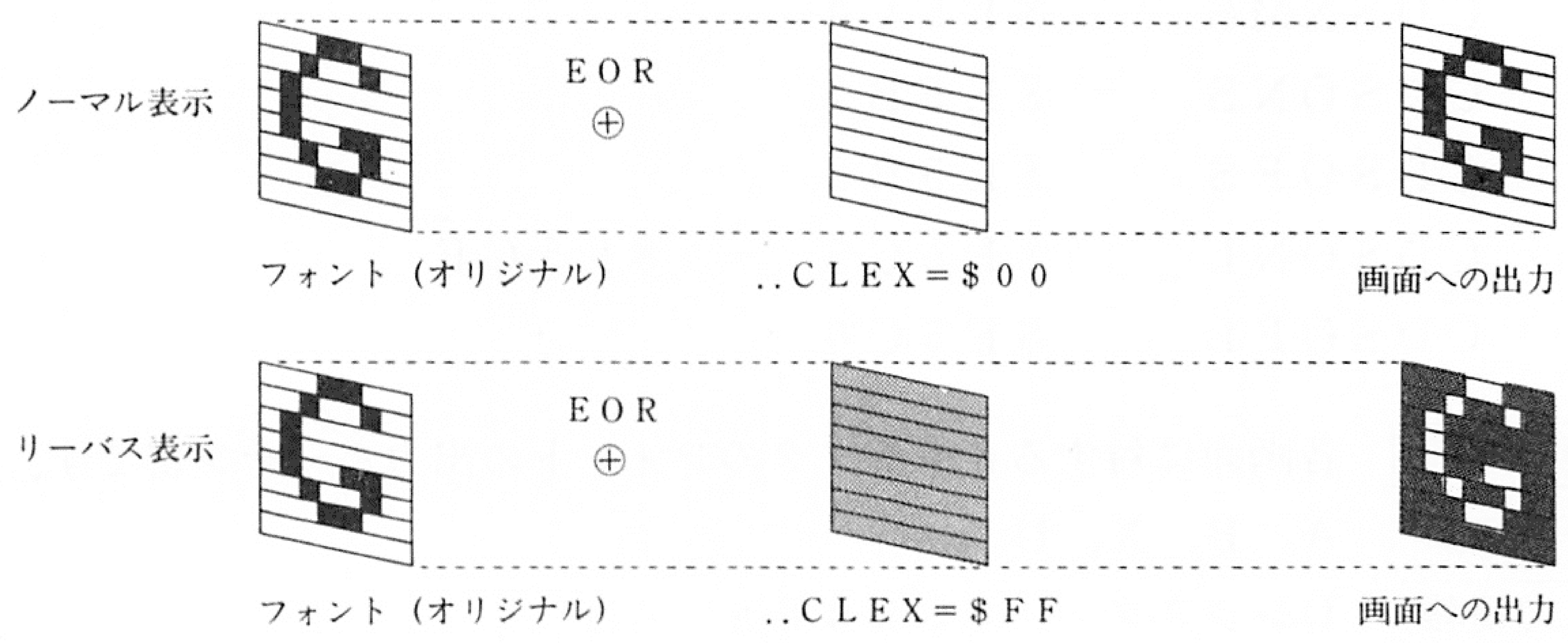
解　説　全体としては，一つの続いたルーチンですが，エントリによって機能が異ります．機能はラベルの４文字目と５文字目によって，また文字サイズの大小はラベルの６文字目によって表現してあり，次のように動作します．

図３・４・30　各エントリの機能



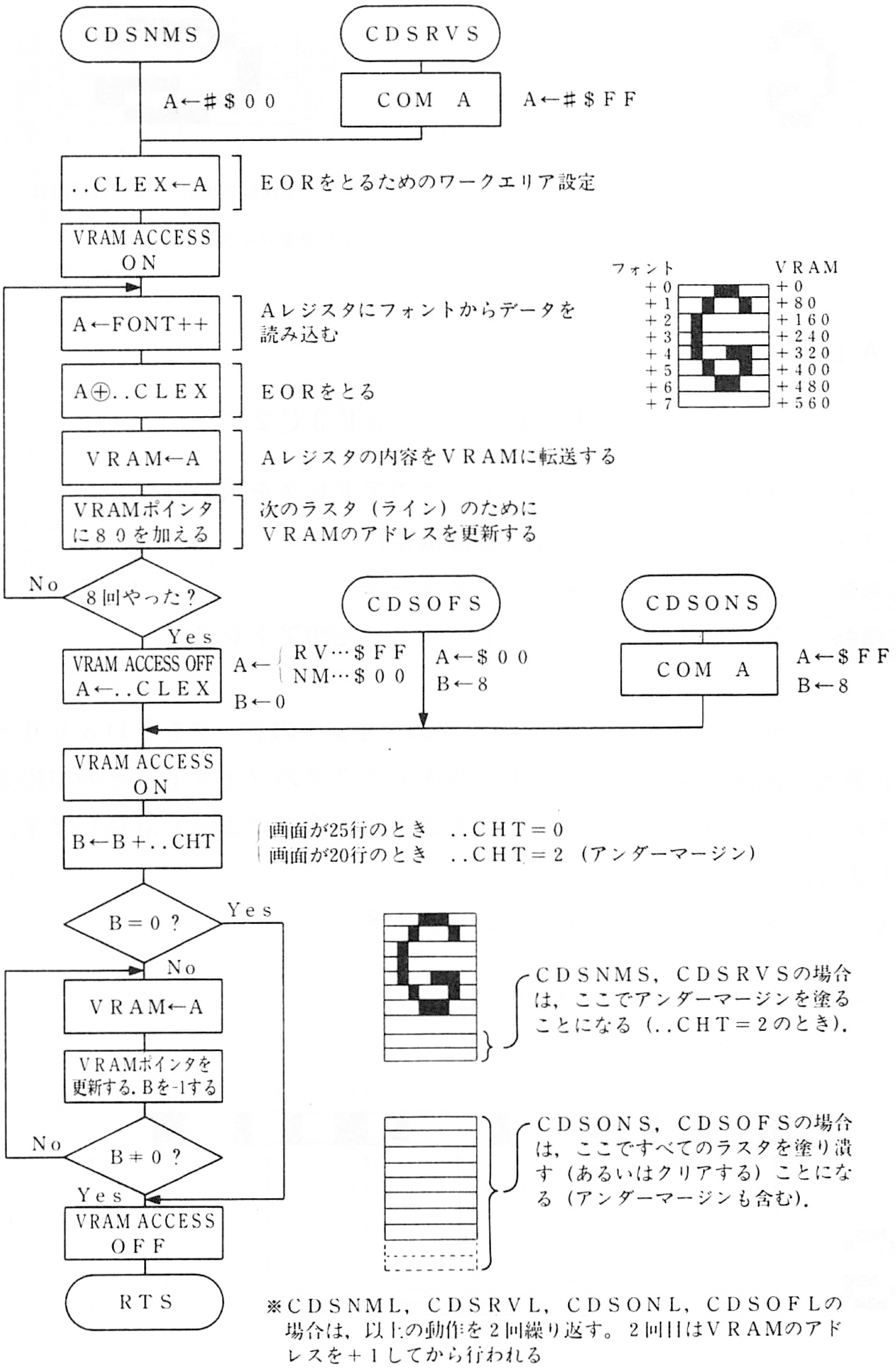
ノーマル表示とリバース表示の場合は，キャラクタフォントの出力の際にワークエリア..ＣＬＥＸとのＥＯＲ（Exclusive ＯＲ）をとって出力されます．..ＣＬＥＸには，リバース表示の場合は＄ＦＦ，ノーマル表示の場合は＄００が代入されます．＄００とＥＯＲをとるときはそのままですが，＄ＦＦとＥＯＲをとるときは各ビットが反転することになり，反転表示をすることになります．

図３・４・31　ノーマル表示とリバース表示



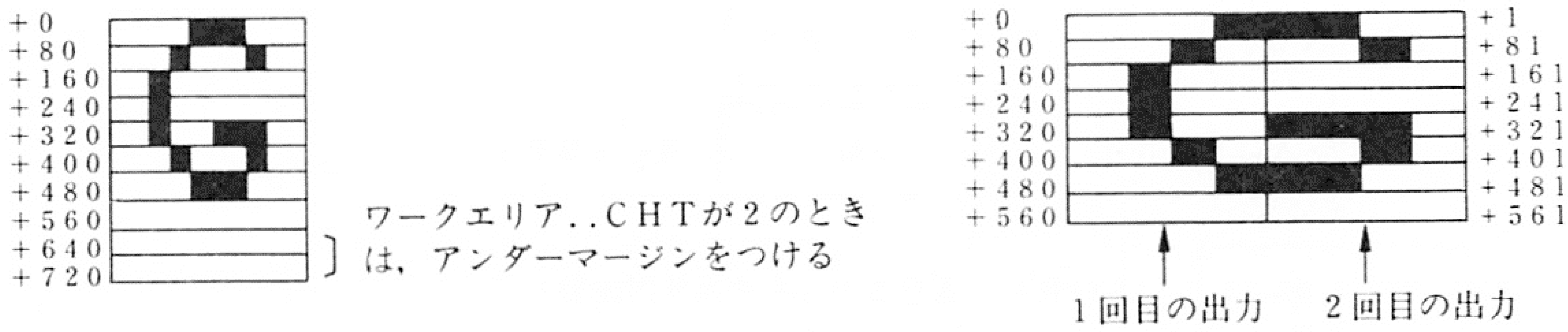
また，ワークエリア..ＣＨＴによってキャラクタのフォントの下にアンダーマージン（余白）を２ライン入れています．アンダーマージンは，機能がＯＦ，ＮＭ（Normal）のときはクリア，ＯＮ，ＲＶ（Reverse）のときは塗りつぶされます．文字のサイズが大のときは，左半分を出力するため，計２回出力を行っています．

図3・4・32　CDSNMS～の処理



※CDSNML，CDSRVL，CDSONL，CDSOFLの場合は，以上の動作を2回繰り返す。2回目はVRAMのアドレスを+1してから行われる

図3・4・33 文字のサイズとアンダーマージン



1）アンダーマージン　　　2）文字サイズ大のとき

CFADR

$E3B7 ～ $E3C2

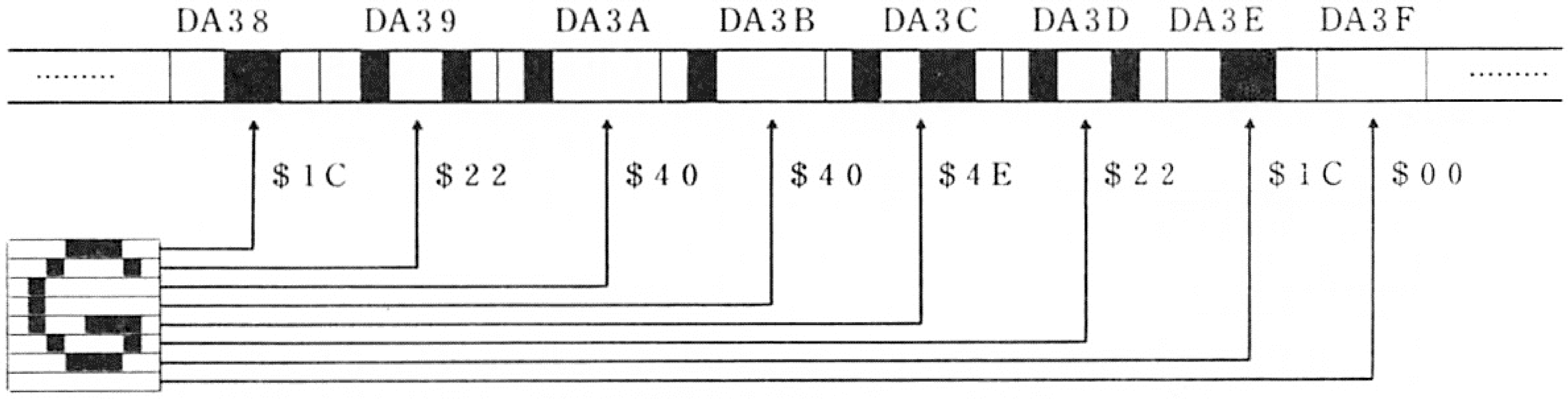
機　能	キャラクタフォントのデータのアドレスを求める.
レジスタ	A，B，X　　（A，Bは保存）
入力情報	Aレジスタ　　キャラクタコード
出力情報	Xレジスタ　　フォントデータの先頭アドレス

解　説　サブシステムは,キャラクタのフォントのデータを＄D800～＄DFFF番地の領域に保持しています．各キャラクタのフォントはその中の8バイトですので，フォントのあるアドレスをこのルーチンによって計算します．
計算方法は次のとおりです．

Xレジスタ＝＄D800＋Aレジスタ＊8

図3・4・34 キャラクタのフォントのデータ

例　大文字Gの場合



ＣＢＡＤＲ

＄Ｅ３Ｃ３　～　＄Ｅ３Ｄ４

機　　能　カーソルのＸ，Ｙ座標（キャラクタ）から，そのバッファアドレスとＶＲＡＭ上のアドレスを求める．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ

入力情報　．ＣＹ　＄Ｄ０５８　カーソルのＹ座標（キャラクタ）
　．ＣＸ　＄Ｄ０５９　カーソルのＸ座標（キャラクタ）

出力情報　．ＣＢＡＤ　＄Ｄ０５Ａ～＄Ｄ０５Ｂ　カーソルの位置に対するキャラクタバッファ内のアドレス（Ｘレジスタ）
　．ＣＡＡＤ　＄Ｄ０５Ｃ～＄Ｄ０５Ｄ　カーソルの位置に対するアトリビュートバッファ内のアドレス(Ｙレジスタ)
　．ＣＶＡＤ　＄Ｄ０５Ｅ～＄Ｄ０５Ｆ　カーソルの位置に対するＶＲＡＭ上のアドレス（Ｄレジスタ）

サブルーチン　ＢＡＤＲ　＄Ｅ２０Ｃ　バッファ内のアドレスを求める
　ＢＶＡＤ　＄Ｅ２２５　ＶＲＡＭ上のアドレスを求める

解　　説　このルーチンは，カーソルのＸ，Ｙ座標が変更された後で，バッファとＶＲＡＭのアドレスをそれに対応づけるために呼び出されます．各ワークエリアは，．ＣＢＡＤと．ＣＡＡＤがＢＡＤＲによって，．ＣＶＡＤがＢＶＡＤによって求められています．これらのワークエリアの内容は，各々Ｘ，Ｙ，Ｄレジスタにも入ることになります．

ＣＡＸＣＲ

＄Ｅ３Ｄ５　～　＄Ｅ３ＤＢ

機　　能　Ｘ，Ｙ座標(キャラクタ)が，範囲内にあるかどうかテストします．

レジスタ　Ａ，Ｂ

入力情報　Ａレジスタ　テストするＸ座標（キャラクタ）
　Ｂレジスタ　テストするＹ座標（キャラクタ）

出力情報　Ｃフラグ　範囲内ならセット（＝１）されます．範囲外ならリセット（＝０）されます．

ワークエリア　．．ＬＮＣ　＄Ｄ０３６　１行の桁数
　．．DMAP＋６　＄Ｄ０４３　（ＰＦキー表示画面開始行）

解　説　ワークエリア..LNC（1行の桁数）とAレジスタ，..DMAP＋6とBレジスタを比較して，レジスタの方が小さければCフラグをセットします．

B I T E X T

$E50B 〜 $E517

機　能　Aレジスタの下位4ビットを拡張して，Bレジスタに格納します．

レジスタ　A，B

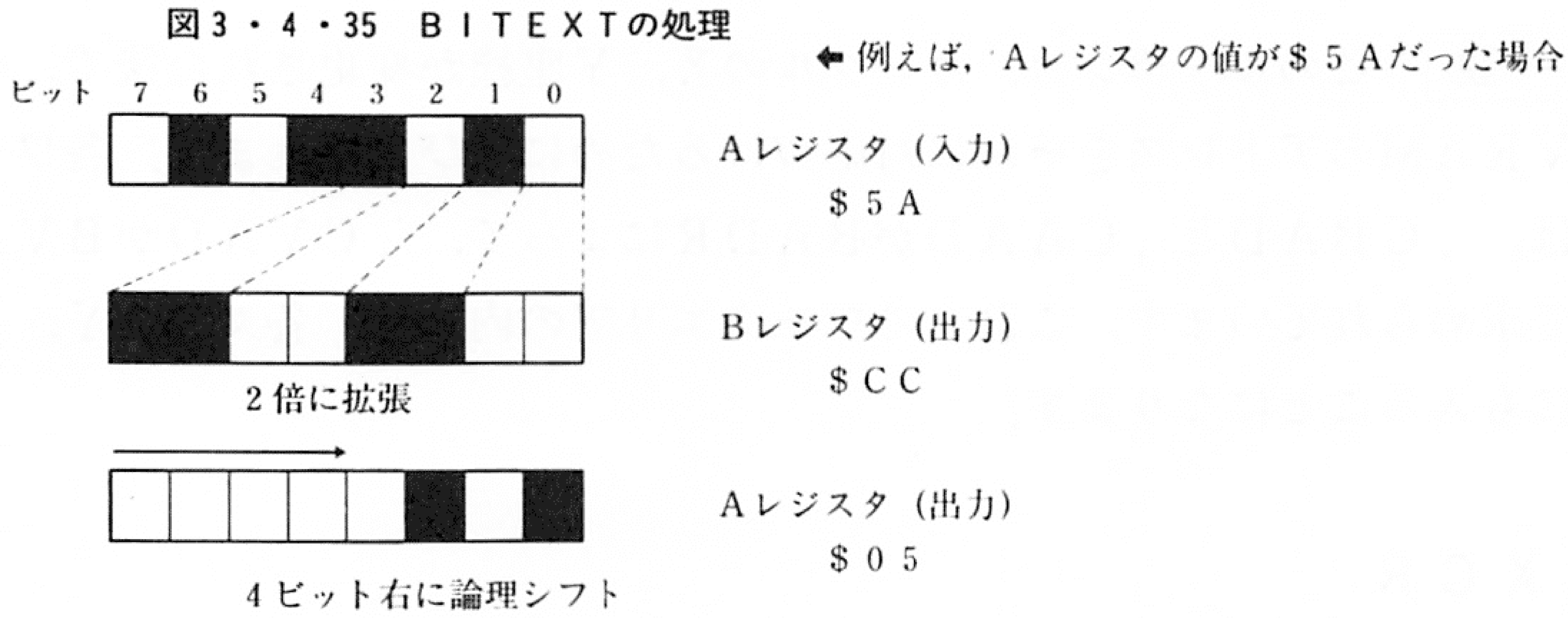
入力情報　Aレジスタ　　拡張するデータ

出力情報　Aレジスタ　　入力したAレジスタを右に4ビット論理シフトしたもの

Bレジスタ　　拡張されたデータ

解　説　図3・4・35のように，Aレジスタのデータを2倍に拡張して，Bレジスタに設定します．このとき，Aレジスタは4ビット右に論理シフトしますので，次にBITEXTを呼びだせば，元のAレジスタの上位4ビットを拡張することになります．例を参照して下さい．

図3・4・35　BITEXTの処理



（※黒い部分が1，白い部分が0を表します）

次の例は，Aレジスタの内容を拡張して，Dレジスタに設定します．

```
LEAS -2,S
JSR  BITEXT
STB  1,S
JSR  BITEXT
STB  0,S
PULS A,B
```

BTOC

BTOC　　　＄E77E　～　＄E792
CTOB　　　＄E787

機　能　バッファポインタのワークエリアの内容をカーソル（の位置）のワークエリアに複写します（BTOC）．あるいはカーソルの位置のワークエリアの内容をバッファポインタのワークエリアに複写します（CTOB）．

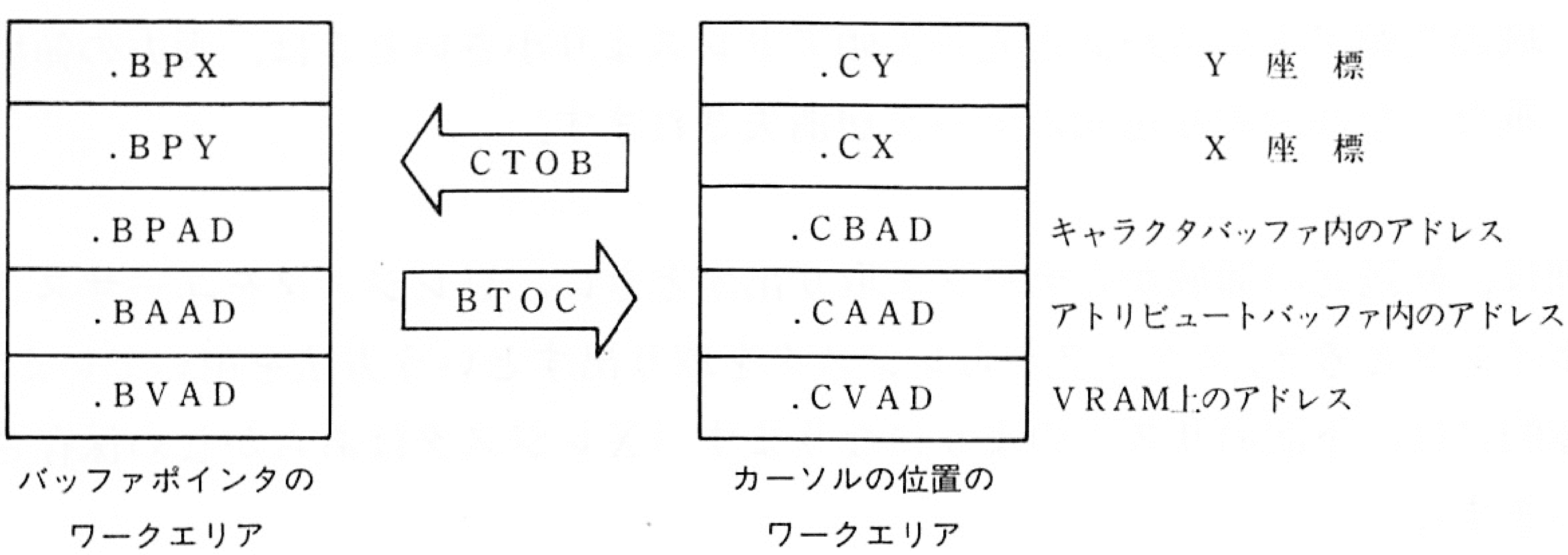
レジスタ　A，B，X，Y

ワークエリア　.BPY　　＄D045～＄D04B（.BPY～.BVAD）
　　　　　　　.CY　　＄D058～＄D05F（.CY～.CVAD）

サブルーチン　MOVE　　＄E44A　ブロック転送ルーチン

解　説　バッファポインタとカーソルの位置のワークエリアの内容を同一のものにします．エントリの違いは，転送方向の違いになります．

図3・4・36　BTOC／CTOBの処理



PUT処理系ではバッファポインタを，GET処理系ではカーソルの位置を中心にして，各処理を行っています．よって一方の処理系のルーチンが，他方の処理系のルーチンをサブルーチンとして使用するとき，ワークエリアの内容を複写してから使用しなければなりません．よって，座標に関するワークエリアを同一のものとするため，このルーチンが用いられます．

ＣＯＰＬＯＰ

＄Ｅ８Ｆ１　～　＄Ｅ９０７

機　　能　ある領域の内容を，別の領域へ転送します．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ　（Ｘは保存）

入力情報　Ｕレジスタ　　転送元の領域の先頭アドレス

Ｘレジスタ　　転送元の領域の最終アドレス＋１

Ｙレジスタ　　転送先の領域の先頭アドレス

解　　説　この転送ルーチンを使用するにあたり，次の点を注意して下さい．

1）Ｕレジスタの内容は，Ｘレジスタの内容より小さくなければなりません．

$$U < X$$

2）８バイトずつ転送していきますので，転送するバイト数は，８で割り切れなければなりません．

$$(X - U) \bmod 8 = 0$$

3）Ｕ＜Ｙ＜Ｘつまり，転送先の領域が転送元の領域と重なり，かつ転送元の領域の先頭アドレスが転送先の先頭アドレスより小さいときは，両方の領域が重なった部分の転送元のデータが消失されます．

処理は，転送元の領域からデータを取り出すときに，Ｕレジスタをユーザスタックポインタと考え，スタックからレジスタを取り出すという方法を用いています．具体的には，下記のリストのようになります（Ｘレジスタはあらかじめ保存されています）．

```
PULU D,X     take 4bytes data from source
STD  0,Y     set them to destination
STX  2,Y
PULU D,X     take 4bytes data from source
STD  4,Y     set them to destination
STX  6,Y
```

次の例は，赤の画面の内容を緑の画面に転送するものです．

```
LDU  #$4000
LDX  #$8000
LDY  #$8000
LDA  VACC
JSR  COPLOP
STA  VACC
```

ＧＥＴＲＮＧ

ＧＥＴＲＮＧ　　$Ｅ９７５ 〜 $Ｅ９８９
ＧＥＴＮＯＰ　　$Ｅ９８９

機　能	現在のウィンドウ画面のコンソールバッファ上での先頭アドレスと終了アドレス（の次のアドレス）を求め，範囲を規定します．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ		
	.ＭＤＲＧ	$Ｄ０４Ｄ〜$Ｄ０４Ｆ	ウィンドウ画面の情報
ワークエリア	ＧＴＴＯＰＣ	$Ｄ０５０〜$Ｄ０５１	キャラクタバッファ上での先頭アドレス
	ＧＴＢＴＭＣ	$Ｄ０５２〜$Ｄ０５３	キャラクタバッファ上での終了アドレス（の次の番地のアドレス）
	ＧＴＴＯＰＡ	$Ｄ０５４〜$Ｄ０５５	アトリビュートバッファ上での先頭アドレス
	ＧＴＢＴＭＡ	$Ｄ０５６〜$Ｄ０５７	アトリビュートバッファ上での終了アドレス（の次の番地のアドレス）
サブルーチン	ＢＡＤＲＬ	$Ｅ２０８	行の先頭のバッファアドレスを求める．

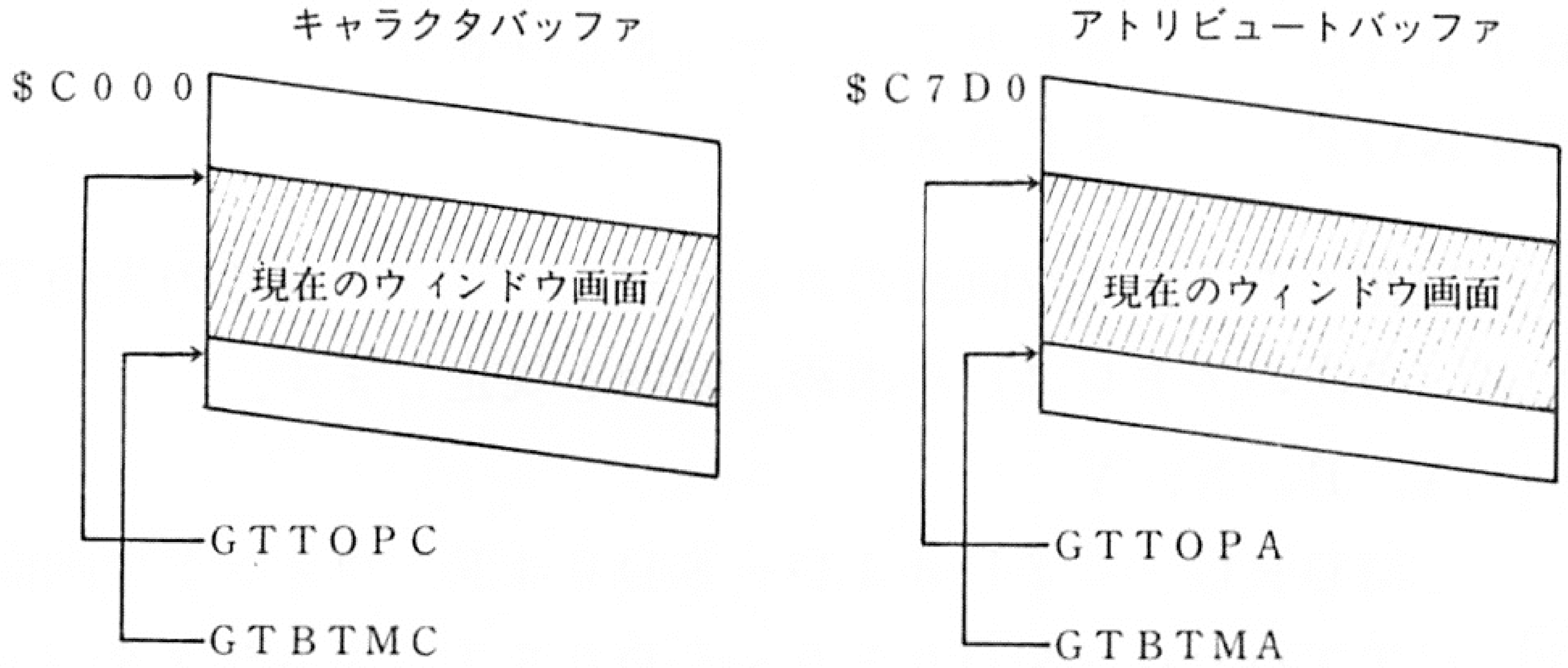
解　説	バッファポインタの位置を変更させて，異る画面に移ったときや，画面の初期化，割当ての変更などが行われたときは，このルーチンが呼ばれ，現在のウィンドウ画面の範囲をワークエリアに規定します．動作は次のようになります．

１）現在のウィンドウ画面の終了行のＹ座標＋１にあたる行の先頭のバッファアドレスをＢＡＤＲＬを使って求め，ＧＴＢＴＭＣとＧＴＢＴＭＡに格納します．

２）同様に，ウィンドウ画面の開始行から，その先頭のバッファアドレスをＧＴＴＯＰＣとＧＴＴＯＰＡに格納します．

なお，ＧＥＴＮＯＰは何の動作もしないオーダコードのエントリで，ＲＴＳ($３９)が書かれています．

図3・4・37 ウィンドウ画面のワークエリア



GETEN

GETENC	$EA79	～ $EA93
GETENA	$EA7F	
GETEN	$EA83	

機　能　バッファアドレスから，XY座標を算出し，その座標にカーソルを設定します．

レジスタ　A，B，X，Y　（Yは保存）

入力情報　Xレジスタ　キャラクタバッファアドレス(GETENCのみ)
Yレジスタ　アトリビュートバッファアドレス(GETENAのみ)
Dレジスタ　ウィンドウ画面内での相対位置(GETENのみ)

ワークエリア　..LNCD　$D035～$D036　1行の桁数（..LNCの手前に0をつけて，2バイトに拡張したもの）
.MDRG　$D04D　ウィンドウ画面の開始行
GTTOPC　$D050～$D051　ウィンドウ画面の先頭のキャラクタバッファアドレス
GTTOPA　$D054～$D055　ウィンドウ画面の先頭のアトリビュートバッファアドレス
.CY　$D058　カーソルのY座標
.CX　$D059　カーソルのX座標

解　説　次のように算出します．

1）GETENCでは，与えられたXレジスタをDレジスタに移し，それからGTTOPCの値だけ引いていきます．

2）GETENAでは，与えられたYレジスタをDレジスタに移し，それからGTTOPAの値だけ引いています．

3）GETENでは，1)または2)で求められたか，あるいは入力情報で与えられたウィンドウ画面での相対位置からXY座標を求めます．求め方は，相対位置を1行の桁数で整数除算し，その商にウィンドウ画面の開始行を足したものをY座標，その余りをX座標とします．

4）3)で求めたY座標が．CYに，X座標が．CXに設定されます．

CURON

$EDD3 ～ $EDE2

機能	カーソルの点滅を開始させます．		
レジスタ	A		
ワークエリア	.CURSR	$D01C	カーソル表示フラグ
	.CURTM	$D01D	カーソルタイマ
	.CURST	$D01E	カーソル状態フラグ

解説　カーソルの点滅を中止していたときは，カーソル表示フラグをセットして再びカーソルを点滅させます．また，カーソルの点滅が禁止されていたときは，カーソルタイマを6にセットし，カーソル状態フラグをクリアしてから，点滅を開始させます．

カーソルは次の3つのモードをとります．そしてこれらのモードを示しているのがカーソル表示フラグ（.CURSR）です．

1）点滅モード　カーソルが一定の周期で点滅しているモードです．
（.CURSR≠0で，.CURSRのビット7＝0）

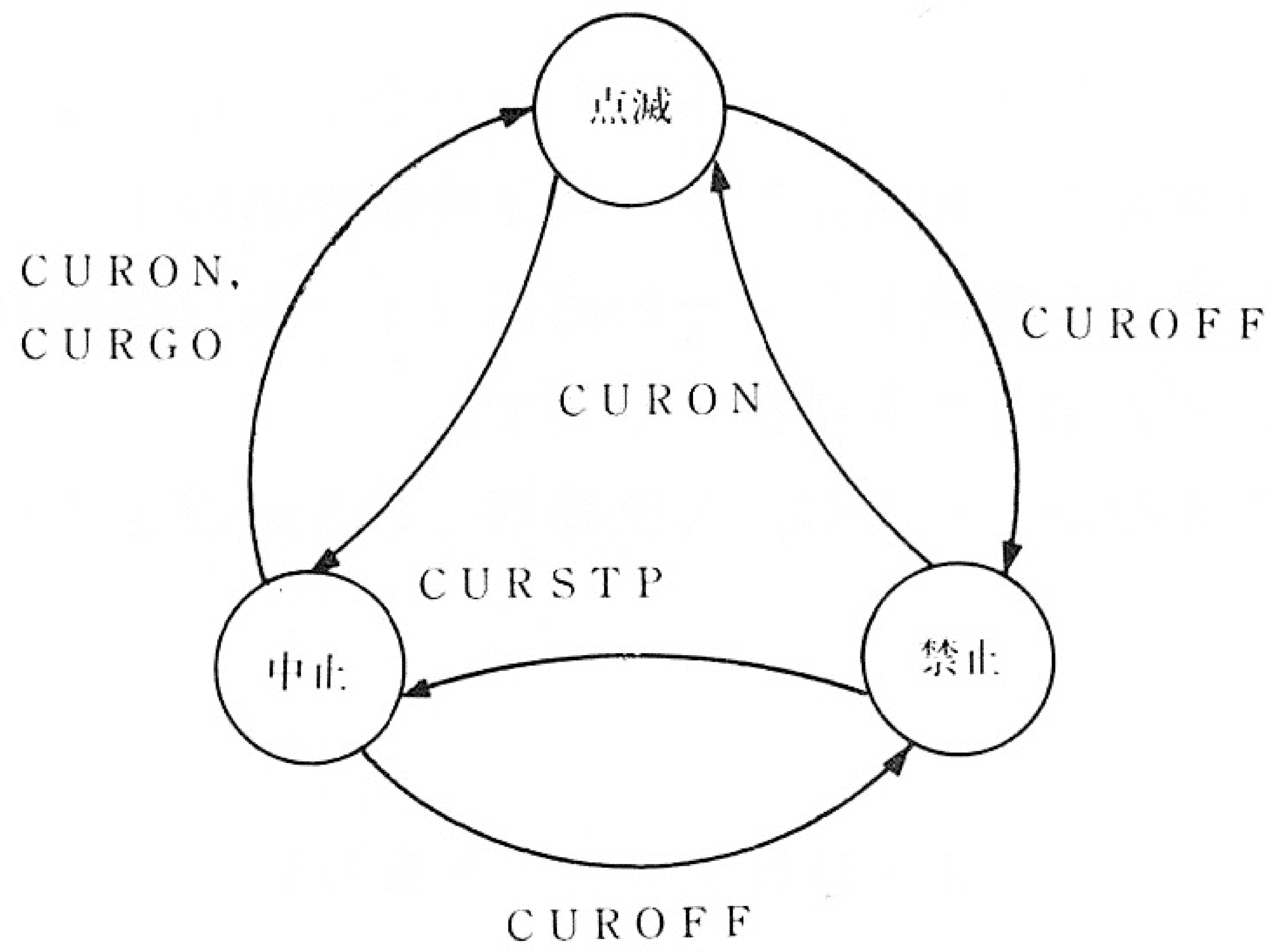
2）中止モード　カーソルの点滅を一時的に中止しているモードです．
（.CURSR≠0で，.CURSRのビット7＝1）

3）禁止モード　カーソルの点滅を禁止しているモードです．
（.CURSR＝0）

各モード間を移行させるのが，CURONよりはじまる一連のルーチンです．図3・4・38に，これらのモードとルーチンとの関係を示しました．

カーソルの点滅は，NMIサービスルーチンで行っています．NMIサービスルーチンでのカーソルの点滅を制御するのが，これらのルーチンの役割です．

図3・4・38　カーソルのモードと点滅制御ルーチン



CUROFF

$EDE3 ～ $EDE7

機　能	カーソルの点滅を禁止させます.
レジスタ	A, B, X, Y, U
ワークエリア	.CURSR　$D01C　カーソル表示フラグ
サブルーチン	CURSCL　$EDED　カーソルのクリア

解　説　カーソルをクリア（CURSCL）した後, カーソル表示フラグをクリアして, カーソルの点滅を禁止させます.

CURSET

$EDE8 ～ $EDEC

機　能	カーソルの位置を設定します.
レジスタ	A, B, X, Y
サブルーチン	BTOC　$E77E　バッファポインタをカーソルへ CURSTP　$EE01　カーソルの点滅を中止する

解　説　カーソルの表示を中止させた後, バッファポインタの位置（を示すワークエリア）をカーソルの位置（を示すワークエリア）にコピーします.

ＣＵＲＳＣＬ

ＣＵＲＳＣＬ　　$ＥＤＥＤ ～ $ＥＥ００
ＣＵＲＣＬＲ　　$ＥＤＥＦ

機　能　カーソルをクリアします．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

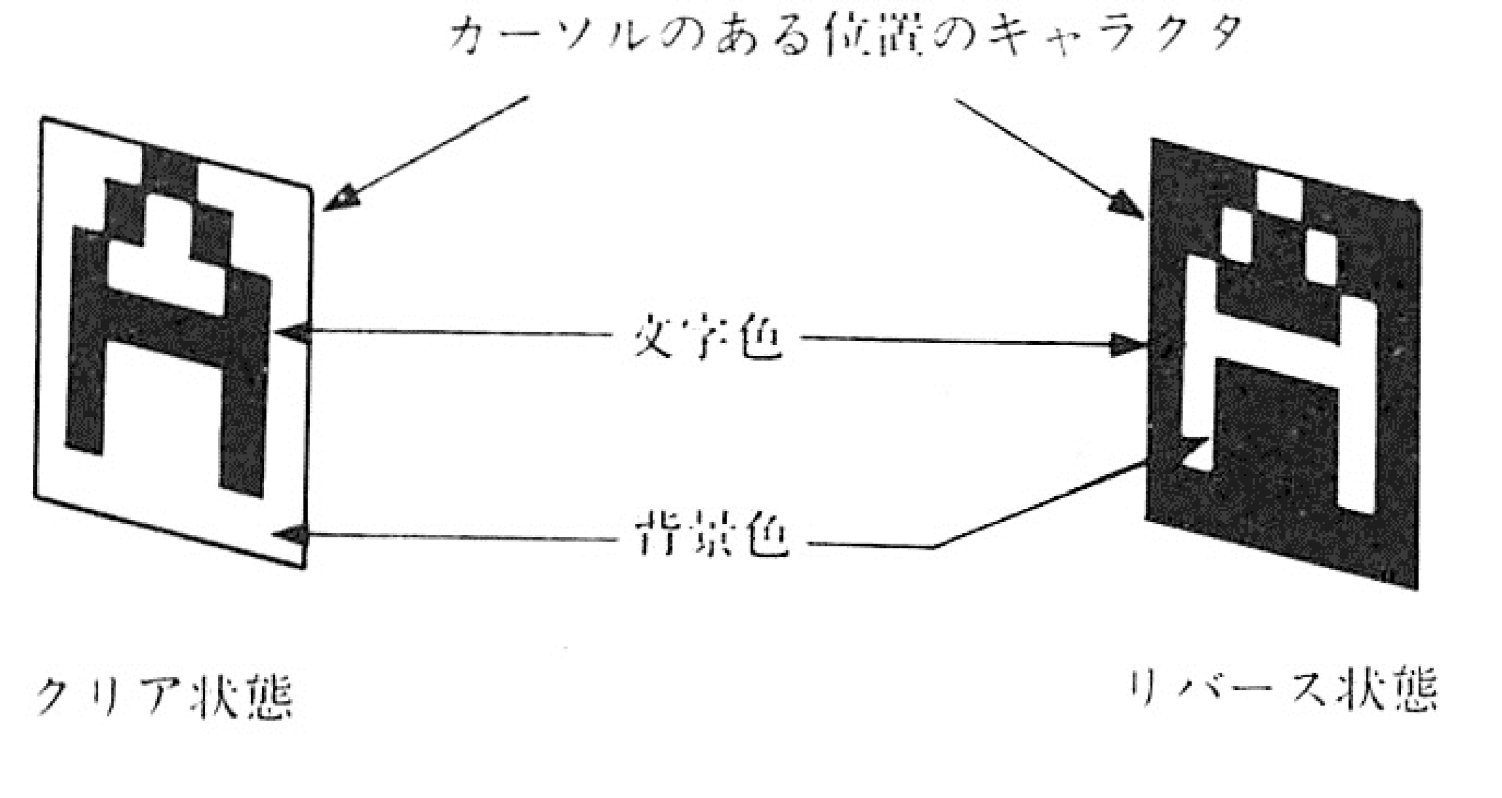
ワークエリア　.ＣＵＲＳＴ　$Ｄ０１Ｅ　カーソル状態フラグ
.ＣＢＡＤ　$Ｄ０５Ａ～$Ｄ０５Ｂ　カーソルのキャラクタバッファアドレス
.ＣＡＡＤ　$Ｄ０５Ｃ～$Ｄ０５Ｄ　カーソルのアトリビュートバッファアドレス

サブルーチン　ＣＤＳＰＣ　$Ｅ５１８　カーソルの位置へキャラクタを表示
ＣＵＲＳＴＰ　$ＥＥ０１　カーソルの点滅を中止する

解　説　エントリがＣＵＲＳＣＬの場合は，カーソルの点滅を中止させますが，ＣＵＲＣＬＲの場合は，カーソルの点滅は中止させません．このルーチンは，カーソル状態フラグをみて，カーソルがリバース状態ならＣＤＳＰＣを用いてカーソルのある位置のキャラクタを元の文字色と背景色で表示します．

ワークエリア .ＣＵＲＳＴ（カーソル状態フラグ）は，カーソルがリバース状態なら$ＦＦ，カーソルがクリア状態なら$００の値を取ります（リバース状態とは，カーソルのある位置のキャラクタの文字色と背景色が反転している状態のことです．クリア状態とは，それらが反転していない状態のことです）．

図3・4・39　カーソルのクリア状態とリバース状態



ＣＵＲＳＴＰ

＄ＥＥ０１　～　＄ＥＥ０Ｄ

機　　能	カーソルの点滅を中止させます．
レジスタ	Ａ
ワークエリア	．ＣＵＲＳＲ　＄Ｄ０１Ｃ　カーソル表示フラグ

解　　説　カーソル表示フラグのビット７をセットして，カーソルの点滅を中止させます．

ＣＵＲＧＯ

＄ＥＥ０７　～　＄ＥＥ０Ｄ

機　　能	中止していたカーソルの点滅を再び開始させます．
レジスタ	Ａ
ワークエリア	．ＣＵＲＳＲ　＄Ｄ０１Ｃ　カーソル表示フラグ

解　　説　カーソル表示フラグのビット７をクリアして，カーソルの点滅を開始させます．

3－5　グラフィック

グラフィックのルーチンにはＬＩＮＥ，ＰＡＩＮＴ，ＰＯＩＮＴ，ＣＨＡＩＮなどがありますが，一つひとつ完全に独立したものではなく，一つの大きな構造を持つルーチンと考えられます．

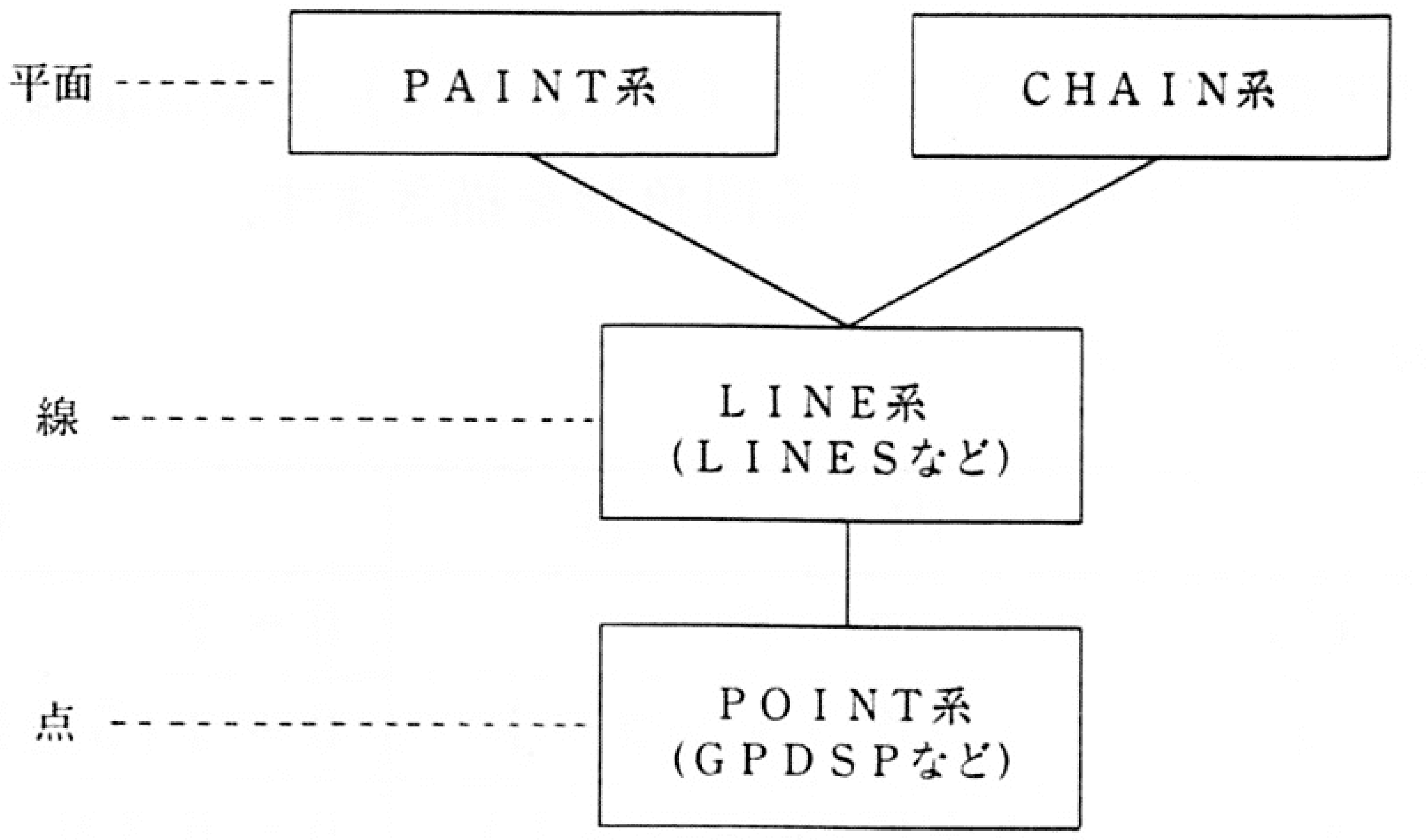
一般にグラフィックルーチンはむずかしいといわれていますが，この原因は主に画面とＶＲＡＭを対応させるのがむずかしいからでしょう．

この項では各ルーチンの具体的解説を行う前に以上のことをふまえて，グラフィックルーチンの構造，ＶＲＡＭと画面との対応について述べておきます．

★　グラフィックルーチンの構造

グラフィックルーチンのおおまかな構造は図３・５・１にあげたようになっています．これは，数学で平面は線で，線は，点で構成されるというのと同じことです．

図3・5・1

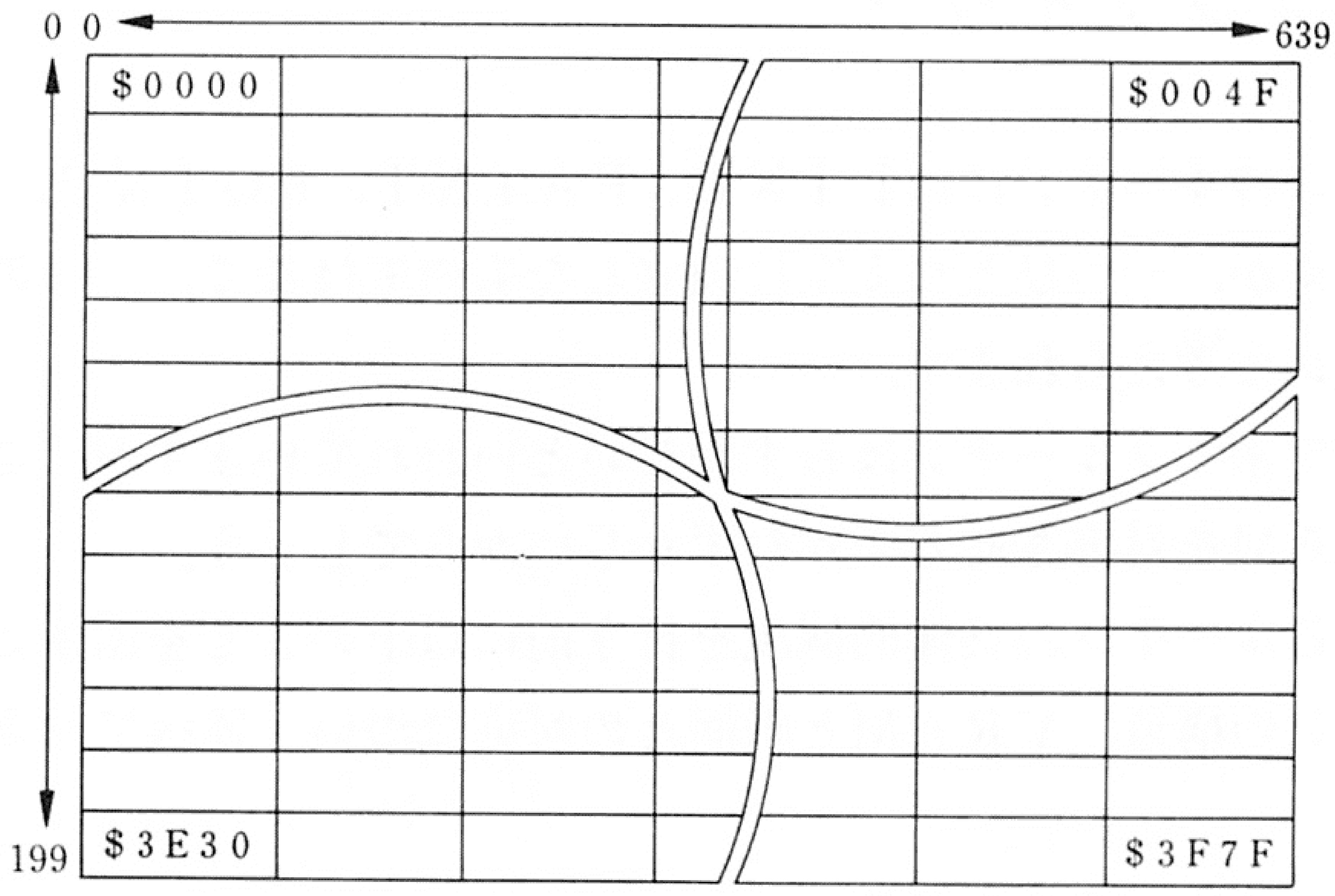


★　画面とＶＲＡＭの対応

画面とＶＲＡＭは図３・５・２のようになっています．画面の１ドットがＶＲＡＭ１ビットに相当し，横ＶＲＡＭ８０バイトで６４０ドットを構成します．つまりＸ＝０，Ｙ＝０は＄００００番地の第７ビット，Ｘ＝６３９，Ｙ＝０は＄００４Ｆ番地の第０ビットに相当します．

このように画面が（６４０，２００）の配列に対し，ＶＲＡＭは（８，１６０００)の配列になっているのがグラフィックをむずかしくしている原因なのです．

図３・５・２



ＶＲＡＭと画面の対応（ＢＬＵＥ）
※ＲＥＤは，わく内のアドレスに＄４０００を，ＧＲＥＥＮは
＄８０００を足す

３－５－１　ＬＩＮＥ　　＄１５　（＄ＥＥ９８）

グラフィック座標（Ｘ１，Ｙ１）と（Ｘ２，Ｙ２）を結ぶ直線，または，（Ｘ１，Ｙ１），（Ｘ２，Ｙ２）を対角線とする四角形を描きます．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
Ｄ３８３	ＣＬ	カラーコード	０～７
Ｄ３８４	Ｆ	ファンクションコード	０～４（５を除く）
Ｄ３８５～Ｄ３８６	Ｘ１	Ｘ１座標（１６ビット）	０～６３９（グラフィック座標）
Ｄ３８７～Ｄ３８８	Ｙ１	Ｙ１座標（１６ビット）	０～１９９（グラフィック座標）
Ｄ３８９～Ｄ３８Ａ	Ｘ２	Ｘ２座標（１６ビット）	０～６３９（グラフィック座標）
Ｄ３８Ｂ～Ｄ３８Ｃ	Ｙ２	Ｙ２座標（１６ビット）	０～１９９（グラフィック座標）
Ｄ３８Ｄ	Ｂ	ボックスフラグ	０＝直線 １＝四角形の枠 ２＝四角形塗りつぶし

主なサブルーチン

$Ｅ４５２	ＩＮＩＴＬＮ	ラインルーチン用イニシァライズルーチン
$Ｅ４４２	ＧＡＸＣＨＫ	グラフィック座標の範囲チェック
$ＥＦ１Ａ	ＬＩＮＥＳ	ライン実行

解　説　ここでは以下のワークエリアを使用しています．

$Ｄ０６０ ～$Ｄ０６１	.ＧＣＬ	カラー，ファンクションコード代入，ＧＰＤＳＰＩ（グラフィック表示用イニシァライズルーチン）の入力情報
$Ｄ０７１ ～$Ｄ０７２	.ＬＸ１	Ｘ１座標
$Ｄ０７３ ～$Ｄ０７４	.ＬＹ１	Ｙ１座標
$Ｄ０７５ ～$Ｄ０７６	.ＬＸ２	Ｘ２座標
$Ｄ０７７ ～$Ｄ０７８	.ＬＹ２	Ｙ２座標

ＬＩＮＥルーチンは，大きく分けると３つに分けられます．一つは直線を描くルーチン，二つ目は四角形を枠を描くルーチン，最後に四角形の塗りつぶしを行うルーチンです．

これらのアルゴリズムを以下に述べます．

1）直線を描くルーチン

このルーチンは，（Ｘ１，Ｙ１），（Ｘ２，Ｙ２）をパラメータとしてＬＩＮＥＳというサブルーチンに代入するだけです．

このＬＩＮＥＳは次の項で解説します．

2）四角形の枠を描くルーチン

このルーチンは図3・5・3のように（Ｘ１，Ｙ１），（Ｘ２，Ｙ２）を対角線とする長方形の他の２つの項点を求めます．（Ｘ２，Ｙ１），（Ｘ１，Ｙ２）がこのように求められます．これをもとに（Ｘ１，Ｙ１）－（Ｘ２，Ｙ１），（Ｘ１，Ｙ１）－（Ｘ１，Ｙ２），（Ｘ２，Ｙ２）－（Ｘ１，Ｙ２），（Ｘ２，Ｙ２）－（Ｘ２，Ｙ１）の順で線を引いていきます．線を引く順序は図３・５・３に書いてあるとおりです．この線を引くのは後述のＬＩＮＥＳというサブルーチンです．

このルーチンは単独でも呼び出し可能でそのアドレスは，＄ＥＥＣＡです．そのときの入力情報は，ＸレジスタにＸ２，あとは共有ＲＡＭ上にＸ１，Ｙ１，Ｘ２，Ｙ２を入れておくこと，＄Ｄ０７１～＄Ｄ０７２にＸ１を入れ＄Ｄ０７３～＄Ｄ０７４にＹ１を入れておくことです．ただしこのルーチンを呼ぶときは，ＩＮＩＴＬＮを呼んでおかなければなりません．さらにデータがグラフィック座標内部にあるかわからないときには，あらかじめＸ１，Ｙ１，Ｘ２，Ｙ２についてＧＡＸＣＨＫルーチンを呼んでチェックしておくことが必要です．

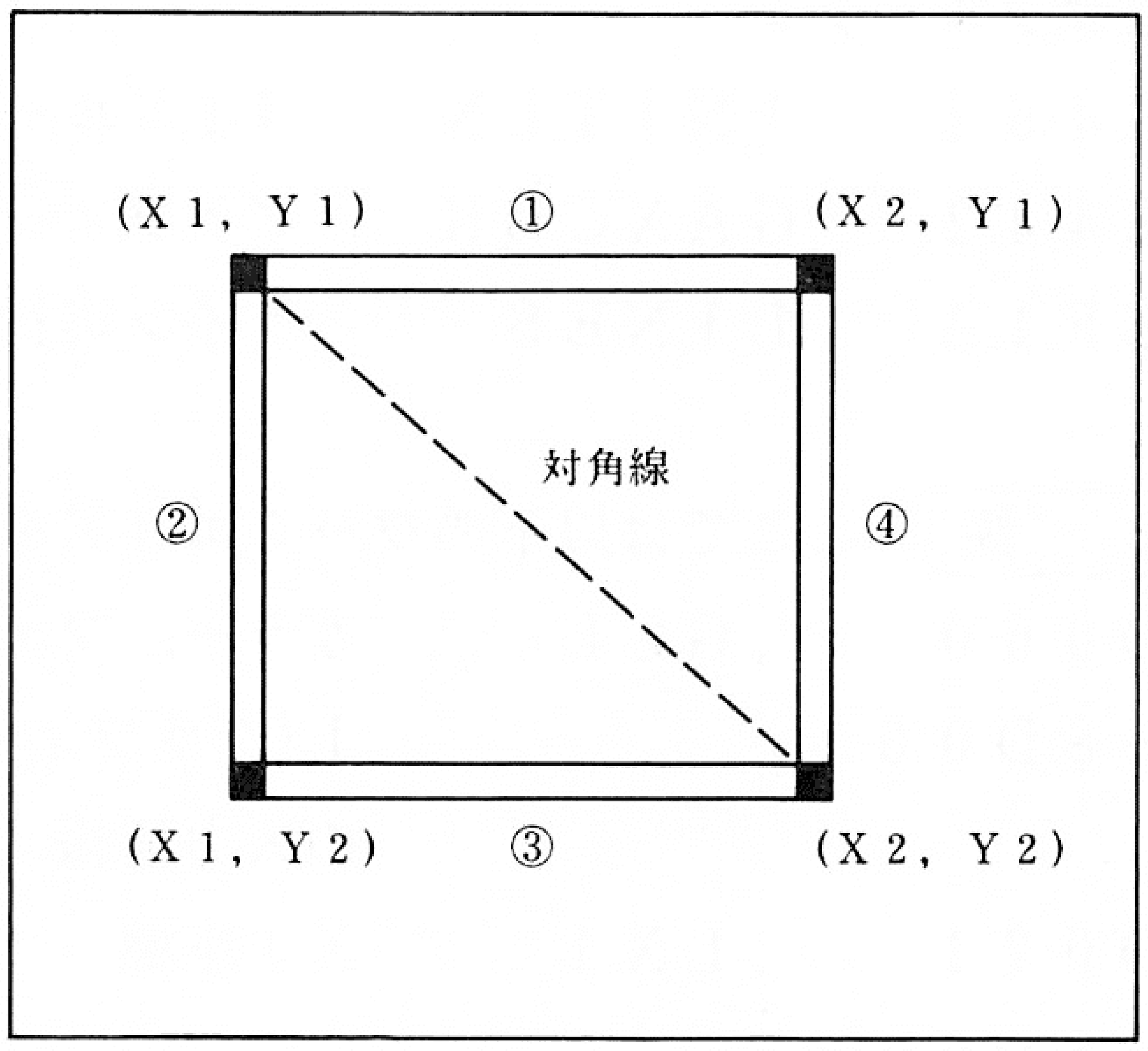


図3・5・3

３）四角形の塗りつぶし

このルーチンは，Ｘ軸に平行な線を｜Ｙ１－Ｙ２｜回実行することによって四角形を塗りつぶします．このアルゴリズムを簡単に説明します．

このルーチンでは以下の２つの状態に分けます．１つは図３・５・４①のＹ２≧Ｙ１のときでもう一つはＹ２＜Ｙ１のときです(②)．まずＹ２≧Ｙ１の場合から説明します．図3・5・4③を参照して下さい．図のように（Ｘ１，Ｙ１）と（Ｘ２，Ｙ１）とを結ぶ線を描きます．次にＹ１を１つ増やして線を今描いた線に描きます．それをＹ１がＹ２になるまで実行します．これで四角形が描けます．Ｙ２＜Ｙ１のときは図３・５・４⑤と図３・５・４⑥のように（Ｘ２，Ｙ２）と（Ｘ１，Ｙ２）とを結ぶ線から描きます．その後Ｙ２を１つ増やしてその下に線を描きます．これをＹ２がＹ１になるまで実行します．

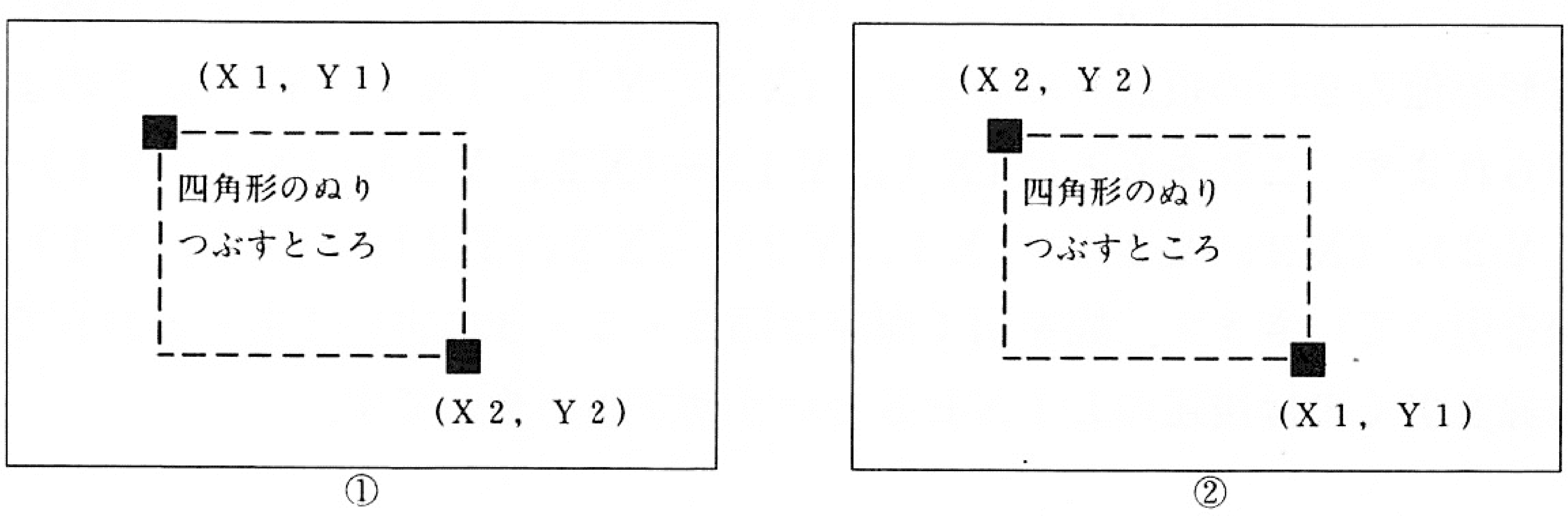


図3・5・4

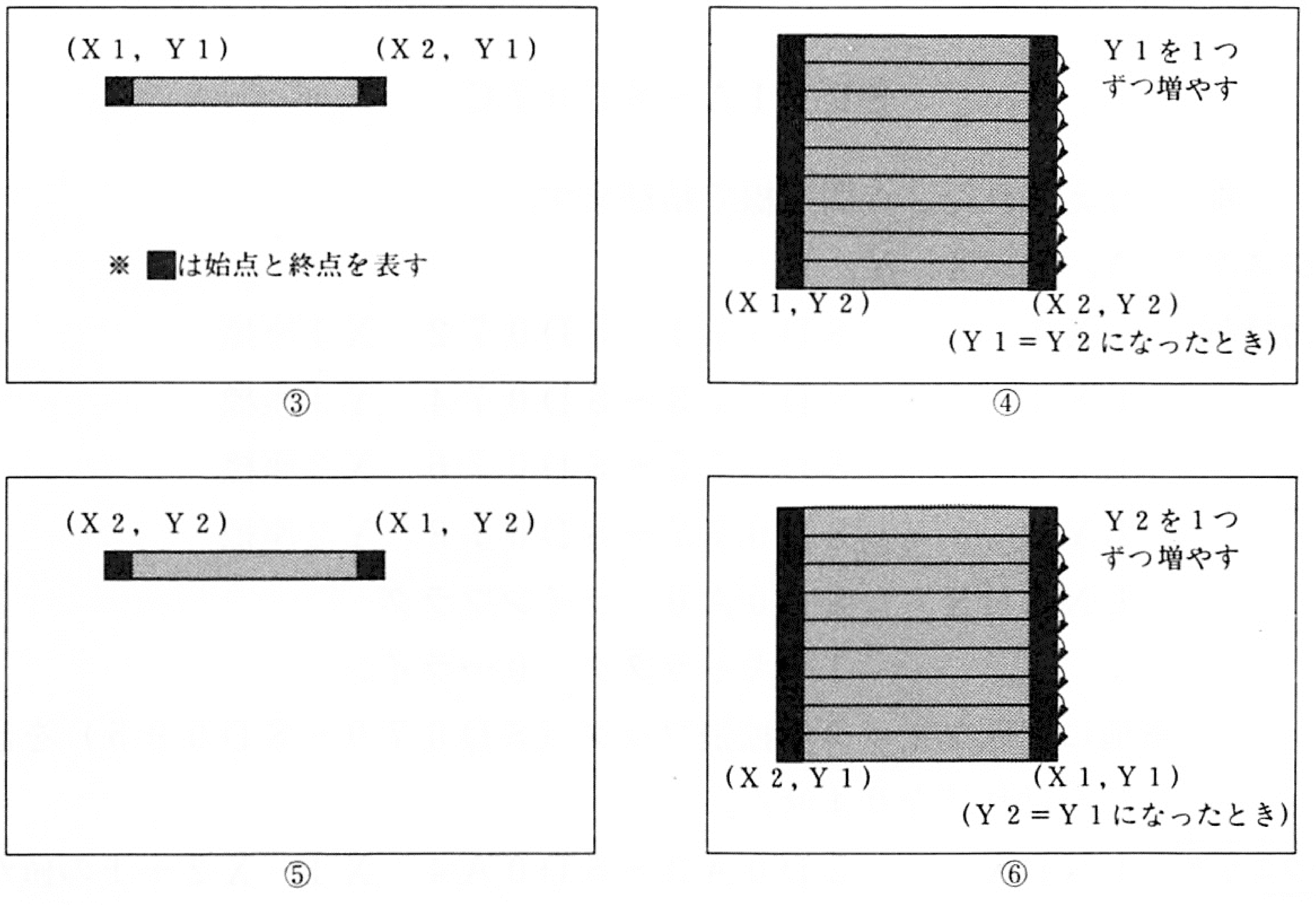


図3・5・4

このルーチンもＬＩＮＥとは独立して利用できます．このルーチンのエントリアドレスは＄ＥＥＦ７番地です．入力情報は＄Ｄ０７１～＄Ｄ０７２にＸ１座標で，＄Ｄ０７３～＄Ｄ０７４にＹ１座標，ＸレジスタにＸ２座標，ＹレジスタにＹ２座標です．このルーチンもＩＮＩＴＬＮ，ＧＡＸＣＨＫを行う必要があります．ただし，このルーチンを呼ぶまえに他のルーチンでＩＮＩＴＬＮを呼んでいて，そのファンクションコードとカラーコードが利用できるときはＩＮＩＴＬＮは不必要でしょう．さらにＸ１，Ｘ２，Ｙ１，Ｙ２がグラフィック座標の外にないと仮定される場合はＧＡＸＣＨＫも不必要でしょう．

これで３つのルーチンについて終ります．次にＬＩＮＥＳというサブルーチンについて述べます．

ＬＩＮＥＳ

＄ＥＦ１Ａ～＄Ｆ０７Ｃ

機　能	与えられた２点間を線で結びます．	
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ	
入力情報	ＬＸ１	＄Ｄ０７１～＄Ｄ０７２　Ｘ１座標
	ＬＹ１	＄Ｄ０７３～＄Ｄ０７４　Ｙ１座標
	ＬＸ２	＄Ｄ０７５～＄Ｄ０７６　Ｘ２座標
	ＬＹ２	＄Ｄ０７７～＄Ｄ０７８　Ｙ２座標
	ＬＮＣＨＲ	＄Ｄ０Ａ０　ラインフラグ １…キャラクタ　０…ライン

※他にグラフィック用拡張フック（＄Ｄ０７９～＄Ｄ０９５）を設定しなければなりません．

ワークエリア	ＬＮＤＸ	＄Ｄ０Ａ３～＄Ｄ０Ａ４　Ｘ１－Ｘ２＋１の値が代入される
	ＬＮＤＹ	＄Ｄ０Ａ５～＄Ｄ０Ａ６　Ｙ１－Ｙ２＋１の値が代入される
	ＬＮＵＸ	＄Ｄ０Ａ７～＄Ｄ０Ａ８　ラインのＸ方向の１ステップ（１ or－１）
	ＬＮＵＹ	＄Ｄ０Ａ９～＄Ｄ０ＡＡ　ラインのＹ方向の１ステップ（１ or－１）
	ＬＮＰＹ	＄Ｄ０ＡＢ～＄Ｄ０ＡＣ　線の傾きを調べる
	.ＧＸ	＄Ｄ０６４～＄Ｄ０６５　実際に点を打つためのＸ座標
	.ＧＹ	＄Ｄ０６２～＄Ｄ０６３　実際に点を打つためのＹ座標
サブルーチン	ＧＰＤＳＰ	＄Ｅ４８Ｆ　指定された点にファンクションコードカラーに合わせて点を打つ
	ＧＰＤＳＰＲ	＄Ｅ４９７　同上
	ＧＡＤＤＲ	＄Ｅ３ＤＣ　Ｘ１，Ｙ１のグラフィック座標からＶＲＡＭアドレスとビットパターンを設定
	ＢＬＵＬＩＮ	＄Ｄ０８Ｃ
	ＲＥＤＬＩＮ	＄Ｄ０８Ｆ
	ＧＲＥＬＩＮ	＄Ｄ０９２　ＩＮＩＴＬＮによりファンクション

BLUSID	$D079	コード，カラーコードによって決定されます．
REDSID	$D07C	
GRESID	$D07F	

解　説　ＬＩＮＥＳルーチンは，グラフィックのラインとキャラクタのラインを引くという，２つの機能がまとまった共用ルーチンですが，ここではグラフィックのラインのみについてふれています（キャラクタラインは３－４－６節を参照して下さい）．このルーチンはＸ１－Ｘ２とＹ１－Ｙ２の値によって次の３つに場合分けして処理を行っています．

1）Ｘ１－Ｘ２＝０（Ｙ軸に平行）
2）Ｙ１－Ｙ２＝０（Ｘ軸に平行）
3）それ以外

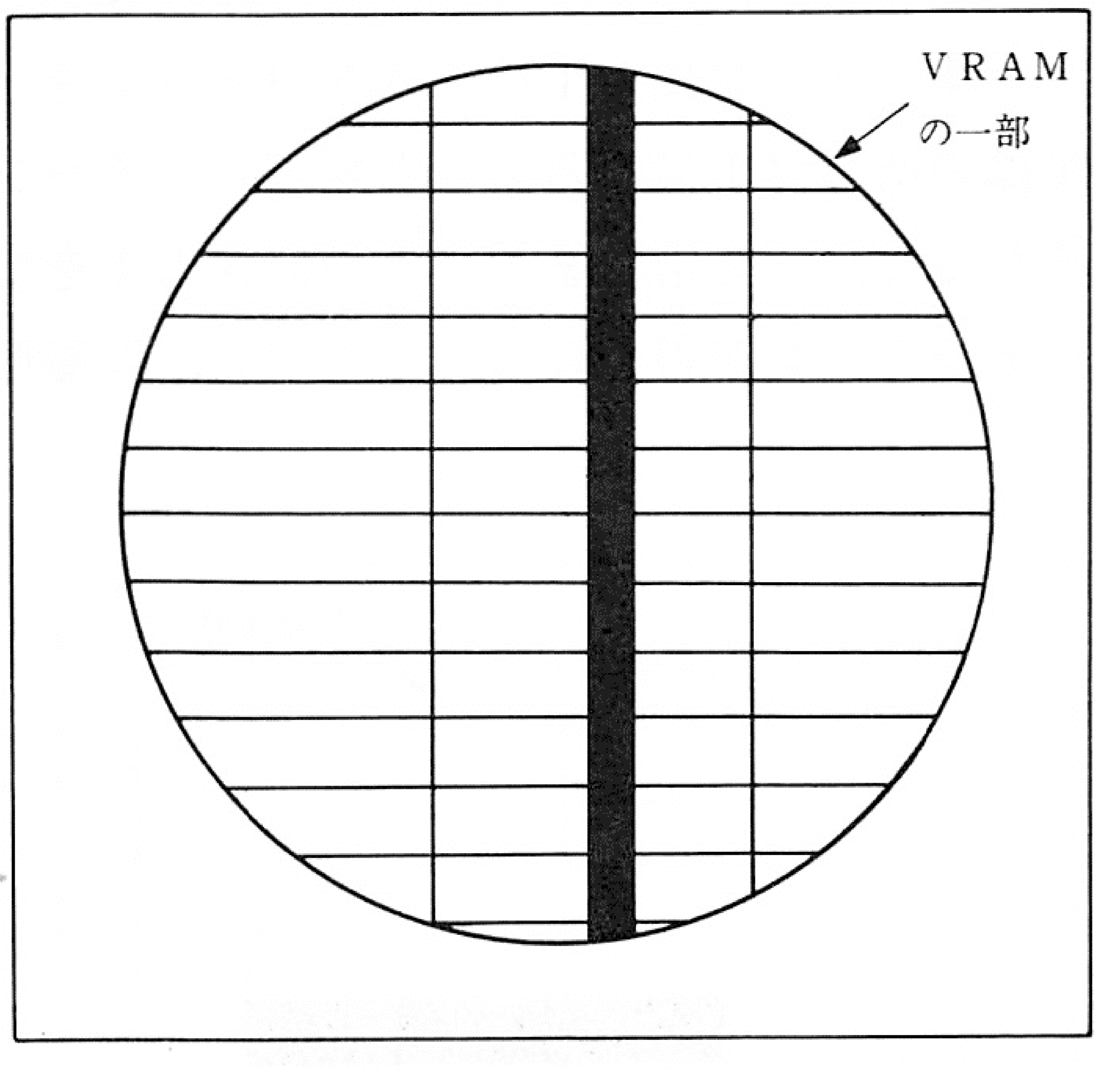


図3・5・5

それでは各々について詳しく述べます．

1）３つの中で最も簡単なものです．

図のようにＹ軸に平行な線を描くとＶＲＡＭに書くビットパターンはすべて同じになり，

00001000

のようになります．ですからビットパターンは，１回だけ求めればよいのです．あとはＶＲＡＭのアドレスですがこれも簡単です．横一列は８０バイトですから，初めに求めたアドレスに８０を足せば次のアドレスが求まります．この操作をＹ１－Ｙ２が０になるまでくり返します．

2）このルーチンはX軸に平行な線を書きます．このルーチンのアルゴリズムを述べます．図３・５・６と図３・５・７をご覧下さい．図３・５・６は最左端の図で図３・５・７は最右端の図です．これを見ればわかるようにX軸に平行な直線のビットパターンは両端を除いてすべて［１１１１１１１１］となっています．ですから両端を除いてすべてのところは，一つのビットパターンで得られるのです．

それでは左側のビットパターンを得る方法を書いてみましょう．この方法は非常に簡単でみごとな方法です．図３・５・８をご覧下さい．左端のグラフィック座標をX１座標とします（X１座標，X２座標の大小でX２座標となることもあります）．X１座標の下位のAND＄０７をとり（X１の８の剰余），この値からX1座標の所だけ１になるようなビットパターンを取り出します（図3・5・9の①）．そしてこれを２倍します．この値から１を引くとBレジスタに左端のバイトのビットパターンが得られるのです（②，③参照）．

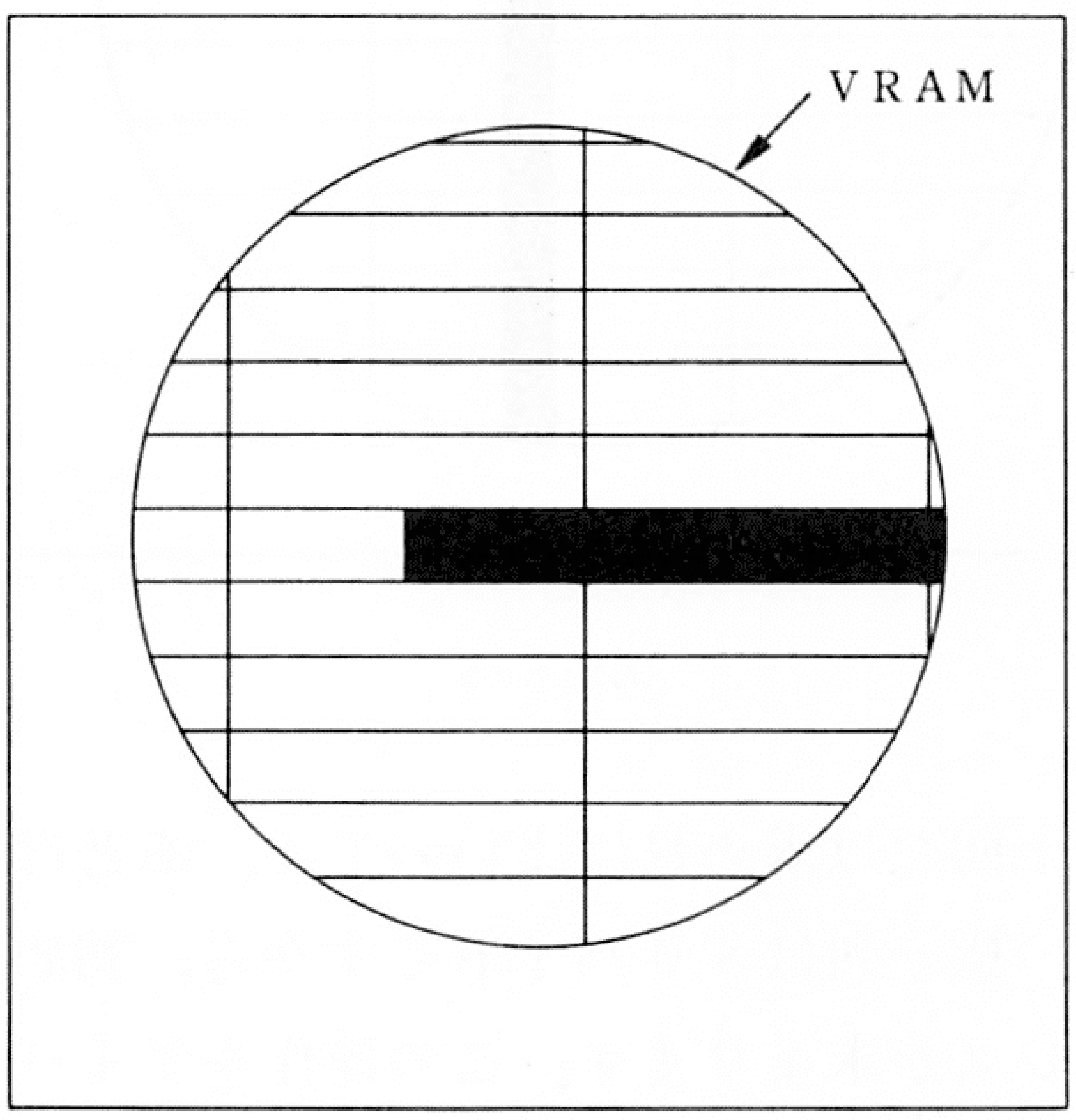


図3・5・6

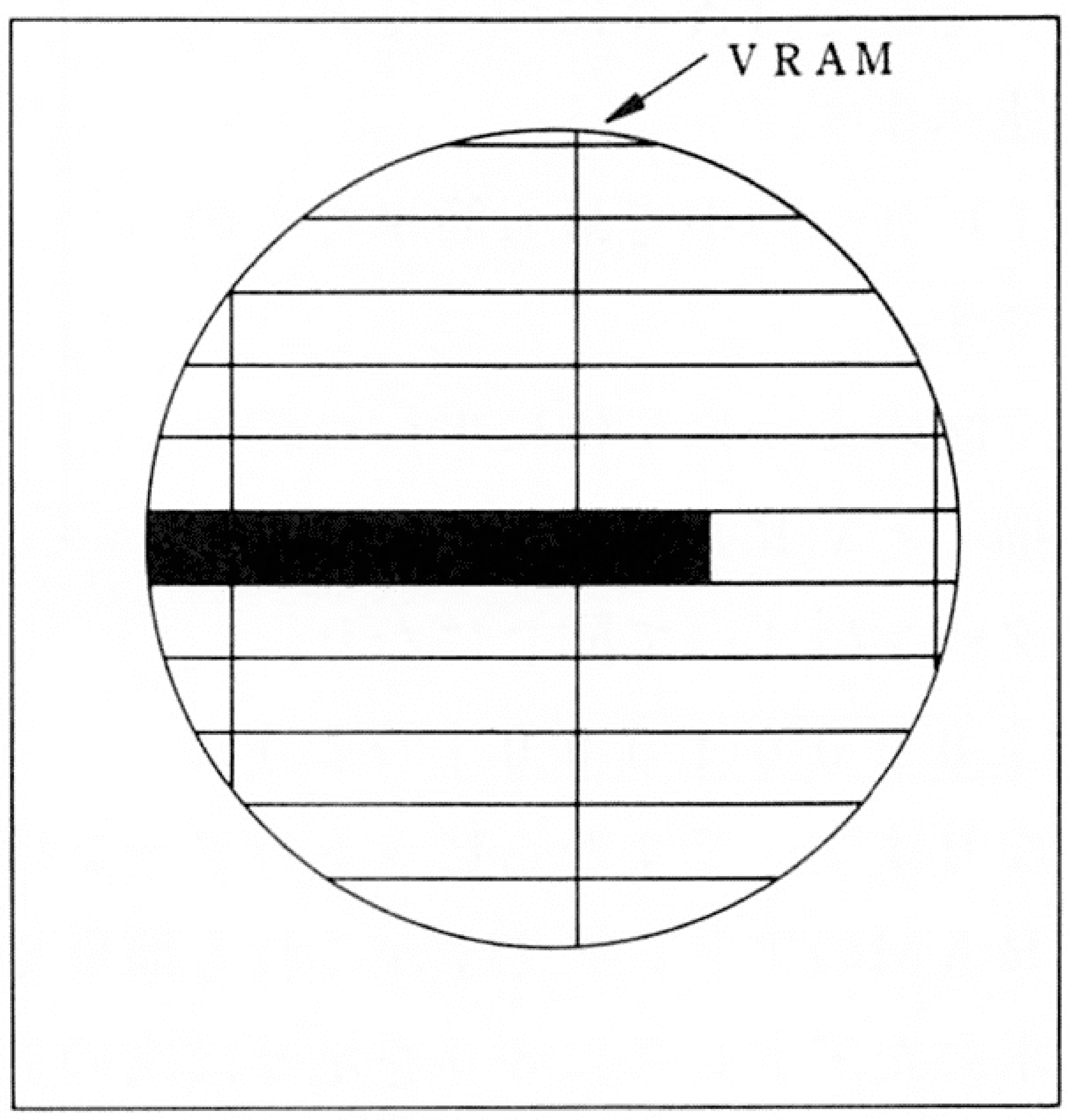


図3・5・7

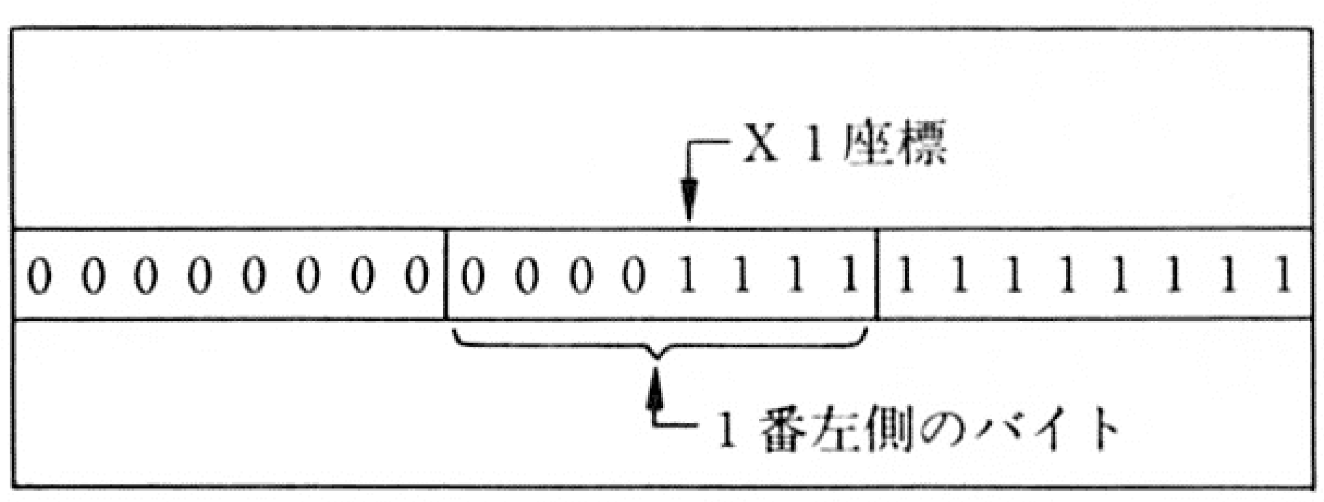


図3・5・8

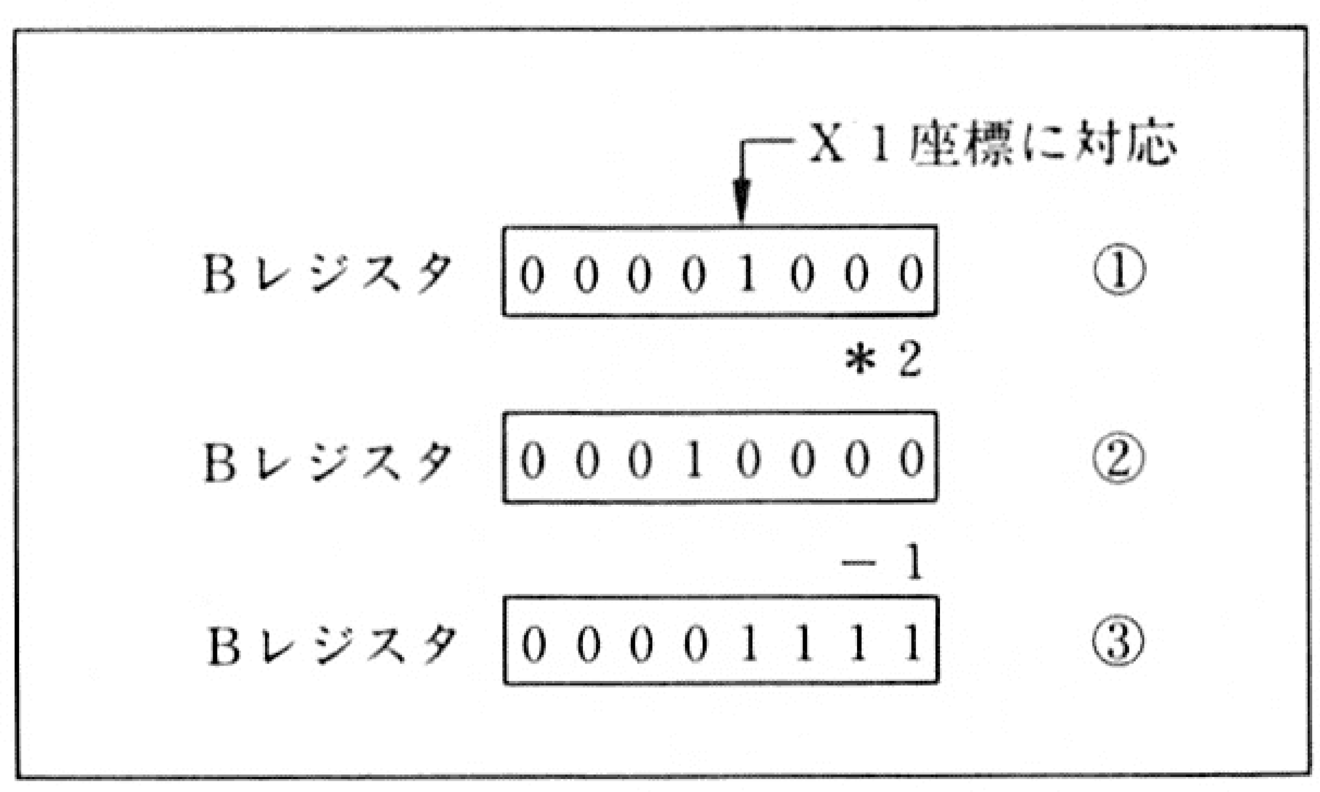


図3・5・9

次に右端のビットパターンを得る方法を述べます．まず左端を求めたときと同様にＸ２座標（Ｘ１座標，Ｘ２座標の大小によってＸ１座標になることもある）からＸ２座標だけ１の立ったビットパターンをＡレジスタに作ります．図３・５・10と図３・５・11の①を参照下さい．その値から１を引いて②の状態にします．この全ビットを反転します．このようにして③のようなビットパターンを得るのです．

これですべてのビットパターンが得られたわけで，このパターンを使いＩＮＩＴＬＮ（＄Ｅ４５２）で作られた拡張フックを利用して線を描きます．

しかしこの方法ではできないことが１つあります．図３・５・12を参照して下さい．このように１バイト中に左端と右端が存在しているときがあります．このときは，Ａレジスタの値とＢレジスタが違う値になってしまい，うまく線が描けません．そこでこの２つのレジスタのＡＮＤを取ります．図３・５・13を参照して下さい．このようにしてＢレジスタにビットパターンを取ります．

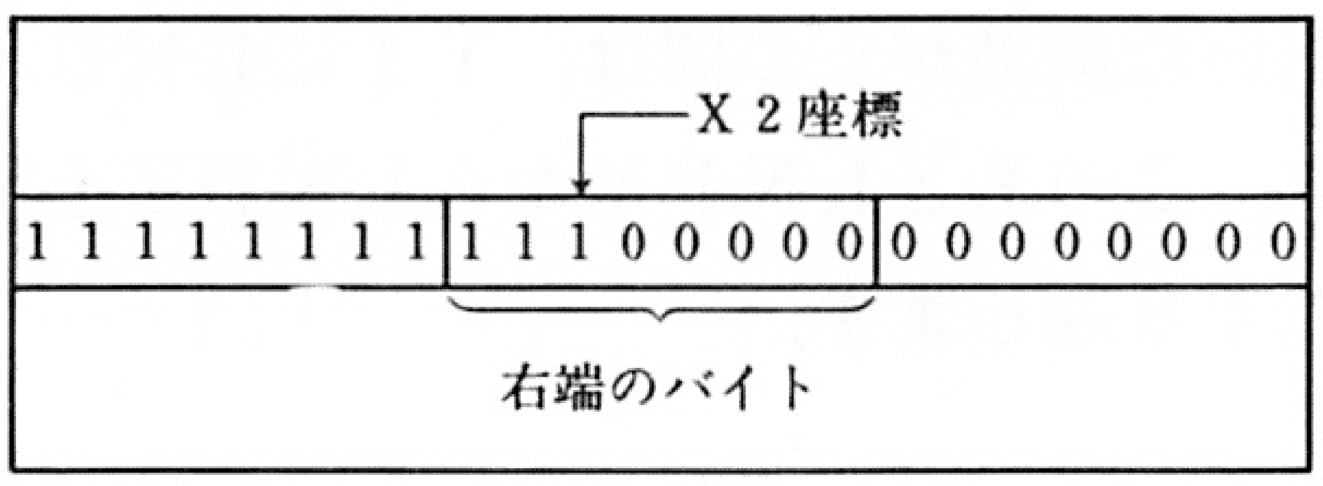


図3・5・10

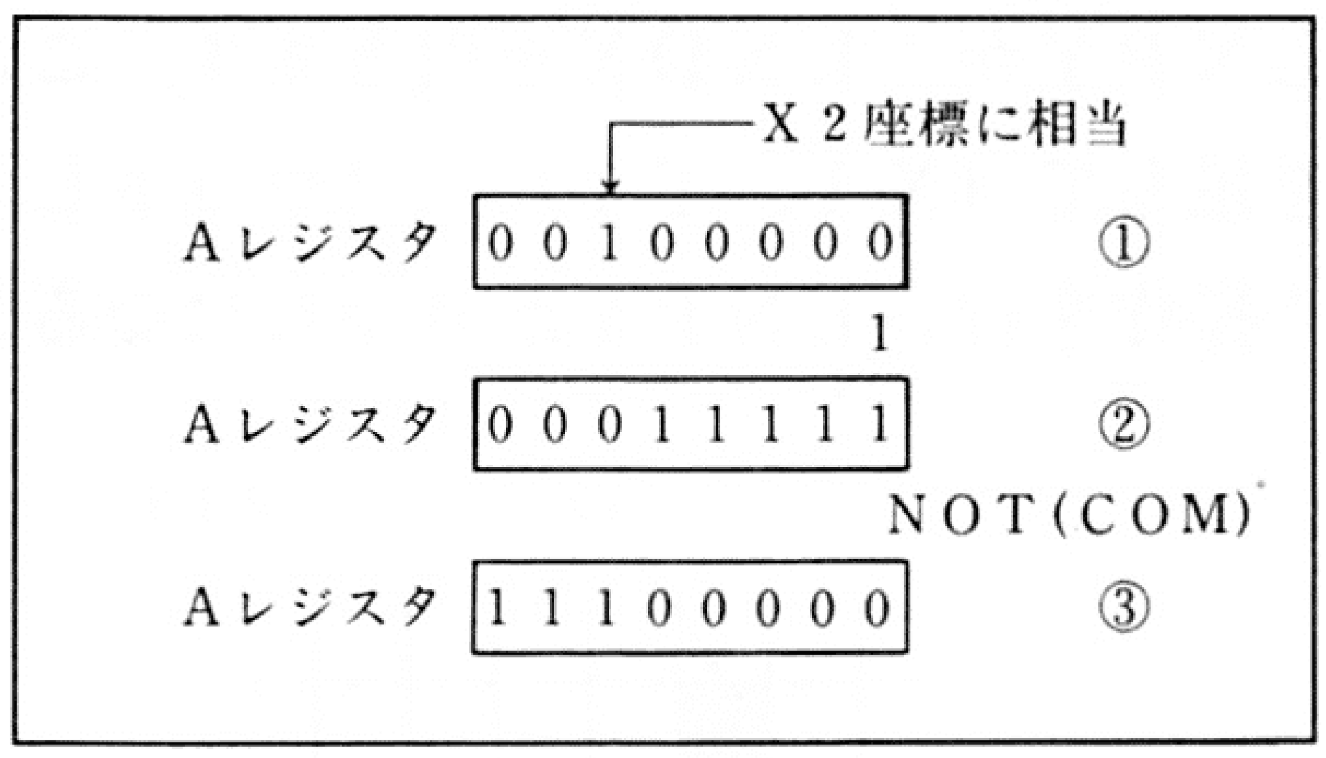


図3・5・11

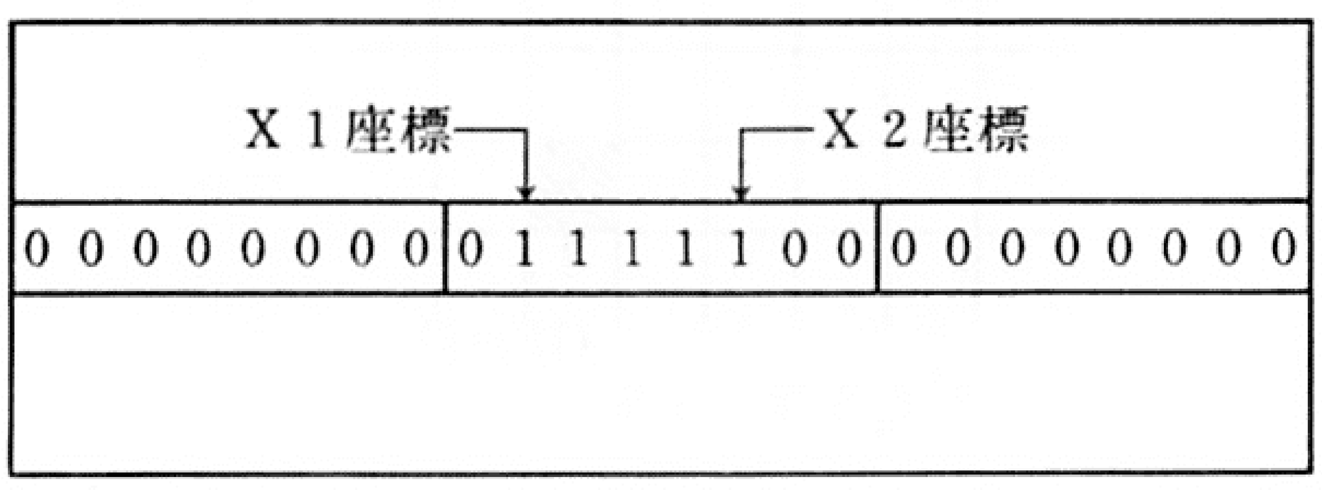


図3・5・12

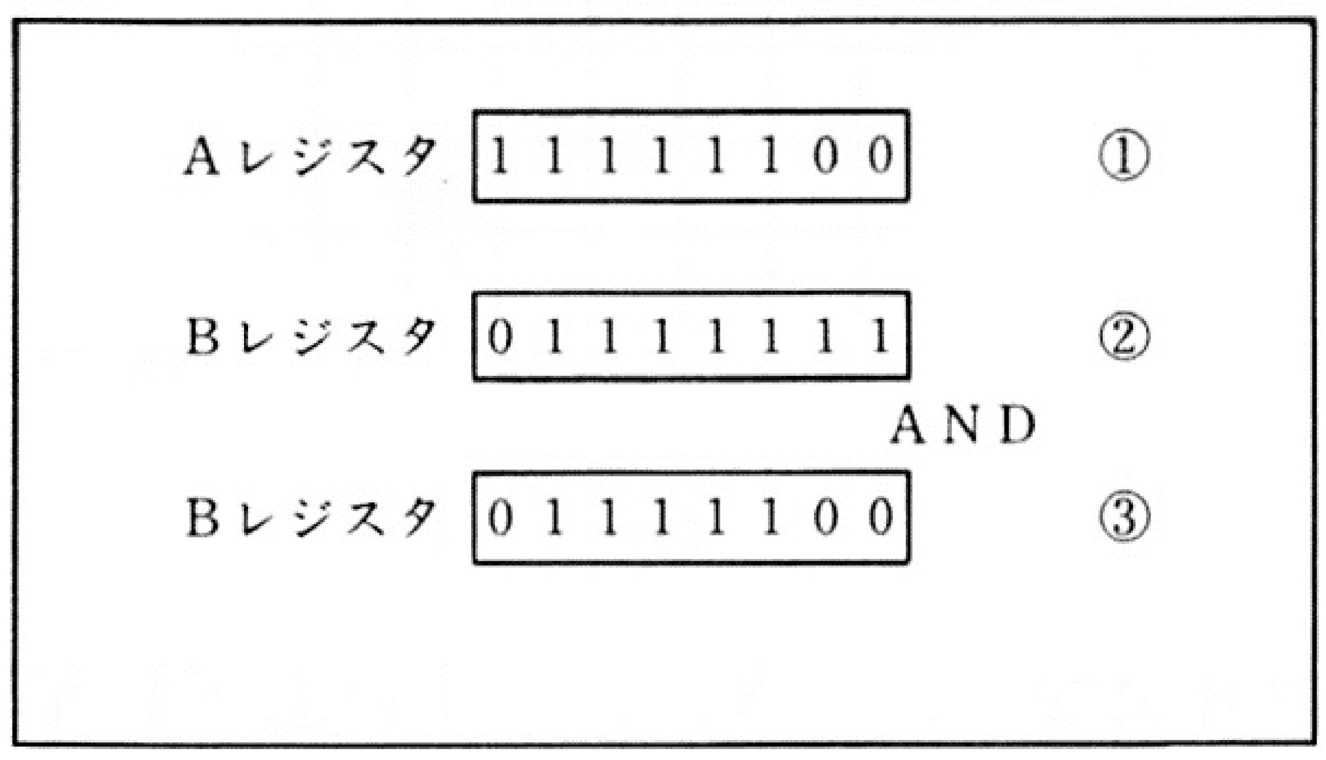


図3・5・13

３）普通に線を描くときは次の式を使うのが最も一般的です．

$$Y=\frac{Y2-Y1}{X2-X1}(X-X1)+Y1$$

もしくは，

$$X=\frac{X2-X1}{Y2-Y1}(Y-Y1)+X1$$

しかしこの２つの式を使ってしまうと傾きを求めるときに小数点以下の値が出

てきてしまいます．ところがこれを浮動小数点を使って処理しようとすると時間がかかってしまいます．そこでこのルーチンでは，この傾きを求めずに点を打つ方法を用いています．

この方法を解説します．具体例として図３・５・14をご覧下さい．
まずあらかじめＬＮＤＸ（Ｄ０Ａ３：Ｄ０Ａ４）に｜Ｘ２－Ｘ１｜＋１を入れ，さらにＬＮＤＹ（Ｄ０Ａ５：Ｄ０Ａ６）に｜Ｙ２－Ｙ１｜＋１を入れます．このとき，ＬＮＵＸ（Ｄ０Ａ７：Ｄ０Ａ８）にＸ２－Ｘ１が負なら＄ＦＦＦＦを，正なら＄０００１を入れます．同様にＬＮＵＹ（Ｄ０Ａ９：Ｄ０ＡＡ）にＹ２－Ｙ１の正負によって値を入れます．この値は，線を描くときの１ステップが＋方向（Ｙ，Ｘが増える方向）か－方向（Ｙ，Ｘが減る方向）かを決める値です．

この例では，ＬＮＤＸ＝4，ＬＮＤＹ＝5，ＬＮＵＸ＝1，ＬＮＵＹ＝1となります．さらにＬＮＰＹ＝0にします．

まず始めに始点をプロットします（図３・５・15）．
ＬＮＰＹにＬＮＤＹを加えます．この例ではＬＮＰＹ＝５となります．このあとＬＮＰＹからＬＮＤＸを引きます．この引いた結果が正の間は，Ｙ１にＬＮＵＹを足して（Ｘ１，Ｙ１）にプロットします．このあとＬＮＰＹからＬＮＤＹを引く操作を行いＬＮＰＹが０か負になるまでくり返します．

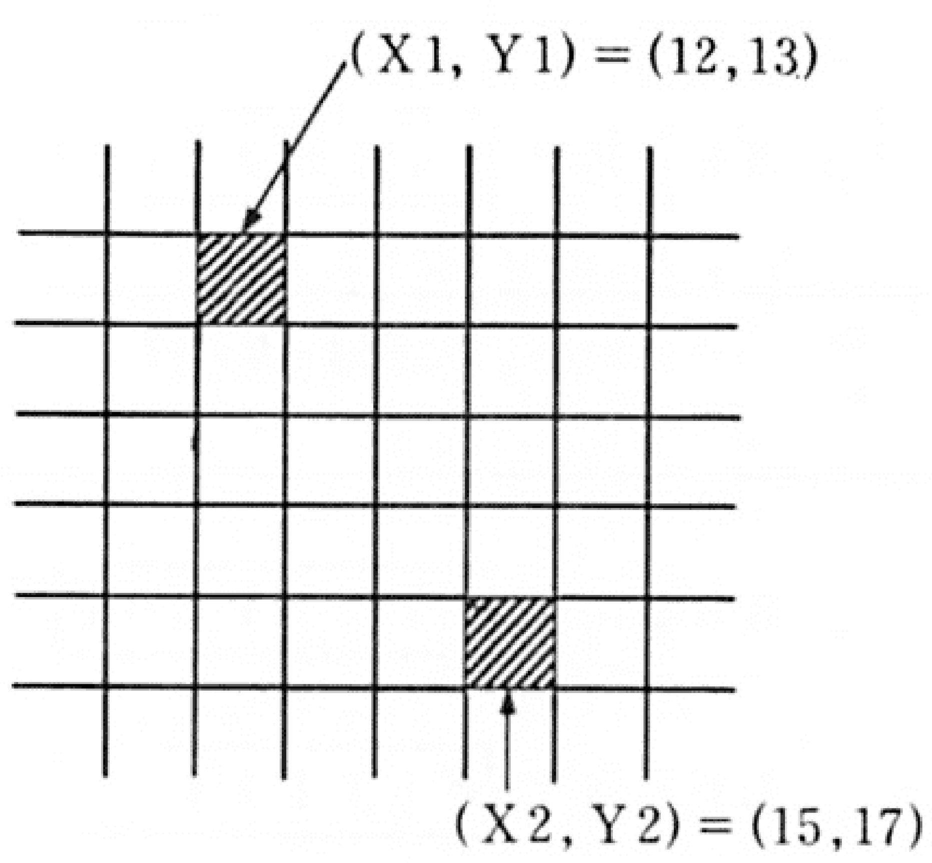


図3・5・14

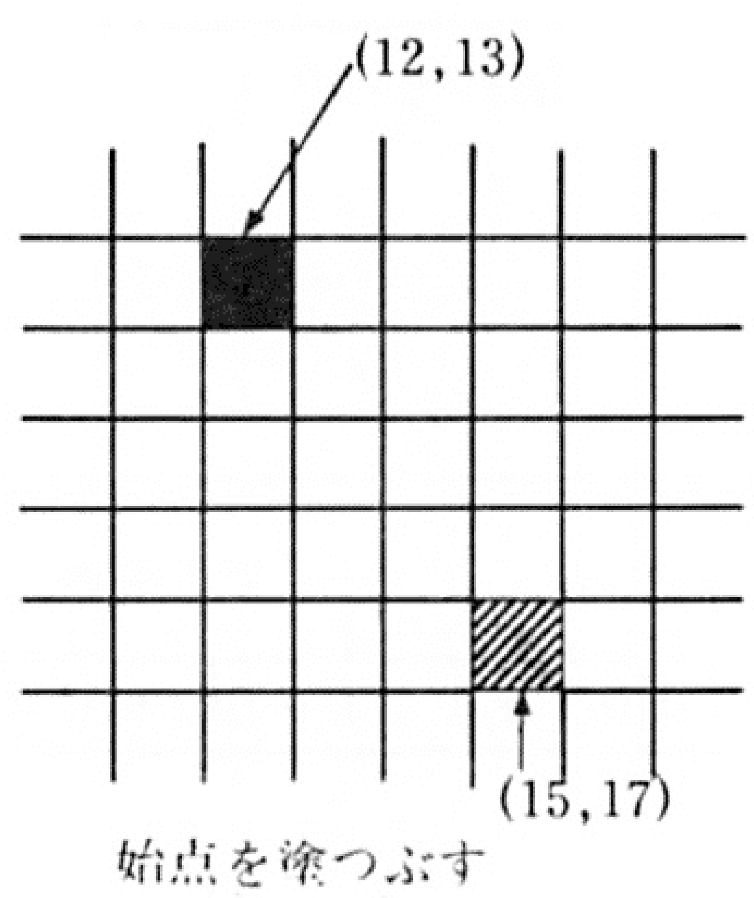


図3・5・15

0になったときは，Ｙ１にＬＮＵＹを加えて，負になったときの操作を行います．

ＬＮＰＹ－ＬＮＤＸが負のときは，ＬＮＰＹにＬＮＤＸを引く前の値を代入します．ＬＮＬＰ＝ＬＮＬＰ－１（ＬＮＬＰは＄Ｄ０ＡＤ～＄Ｄ０ＡＣの内容でＬＮＤＸの内容が初期設定されています）を実行しこの値が0になったときはルーチンを終了させ，他のときは（Ｘ１，Ｙ１）に点を打ってＬＮＰＹにＬＮＤＹを加えるところからくり返します．

例を使って説明します．ＬＮＰＹからＬＮＤＸを引くとＬＮＰＹ＝５－４＝１＞

０なので図３・５・16のように（１２，１４）に点を打ちます．さらにＬＮＰＹからＬＮＤＸを引きます．ＬＮＰＹ＝１－４＝－３＜０なのでＸ１に＋１を足します．ＬＮＬＰの内容はＬＮＬＰ＝３となります．またＬＮＰＹは１のままです．（１３，１４）に点を打ちます（図３・５・17)．ＬＮＰＹにＬＮＤＹを加えます．ＬＮＰＹ＝１＋５＝６．ＬＮＰＹからＬＮＤＸを引きます．ＬＮＰＹ＝６－４＝２＞０なのでＹ１を＋１して（１３，１５）に点を打ちます．ＬＮＰＹからさらにＬＮＤＸを引きます．ＬＮＰＹ＝２－４＝－２＜０なので，ＬＮＰＹは２のままにします．ＬＮＬＰ＝ＬＮＬＰ－１＝２となります．Ｘ１を＋１します．ですから（１４，１５）に図３・５・19のように点を打ちます．ＬＮＰＹにＬＮＤＹを加えます．ＬＮＰＹ＝２＋５＝７となります．この値からＬＮＤＸを引きます．ＬＮＰＹ＝７－４＝３＞０なのでＹ１を＋１して（１４，１６）に図３・５・20のように点を打ちます．ＬＮＰＹからＬＮＤＸを引きます．ＬＮＰＹ＝３－４＝－１＜０なので，ＬＮＰＹは３のままで，ＬＮＬＰ＝ＬＮＬＰ－１＝１となります．Ｘ１に＋１して（１５，１６）に点を打ちます（図３・５・21)．さらに続けて，ＬＮＰＹにＬＮＤＹを加えて，ＬＮＤＸを引きます．ＬＮＰＹ＝３＋５－４＝４＞０なので，Ｙ１に＋１して（１５，１７）に点を打ちます．ＬＮＰＹからＬＮＤＸを引き，ＬＮＰＹ＝４－４＝０なので，Ｙ１に＋１を加えますが，ＬＮＬＰ＝１－１＝０となって，ここでルーチンは終了します．

これでＬＩＮＥＳのルーチンはすべて終りなのですが，実は３）のルーチンだけでも十分線が描けるのです．しかし時間が相当かかるため，このような３つのルーチンを作ったと予想されます．ＦＭ－８Version３．３では３）のルーチンだけなのです．

以下にＢＡＳＩＣで作った３）についてのシミュレーションをフローチャートのかわりに載せておきます．

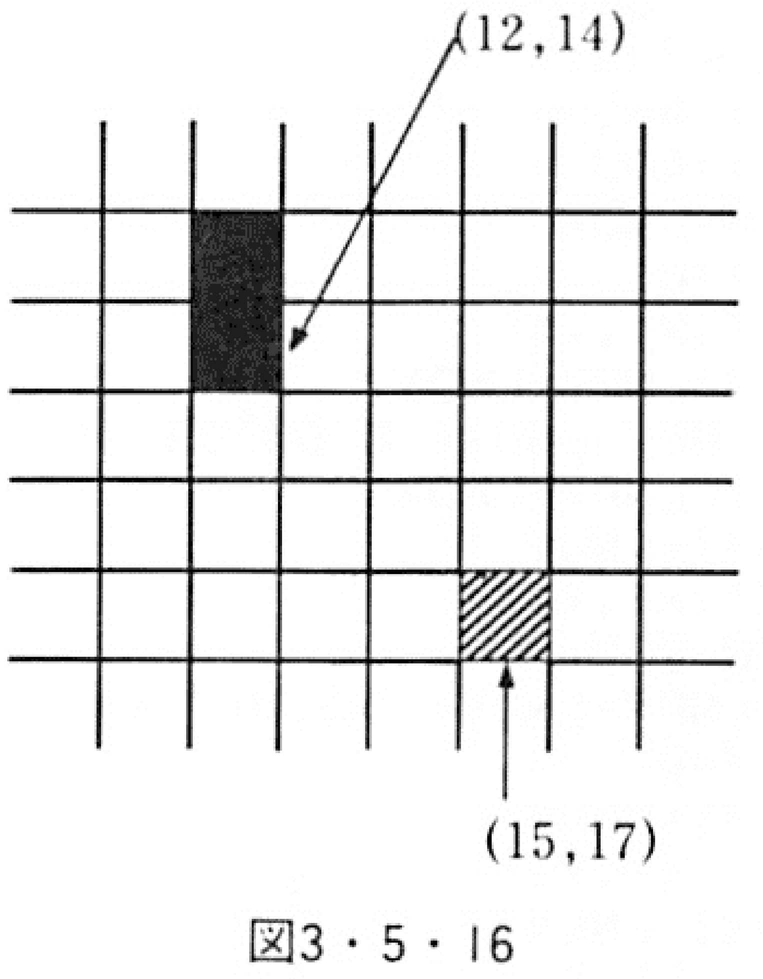


図3・5・16

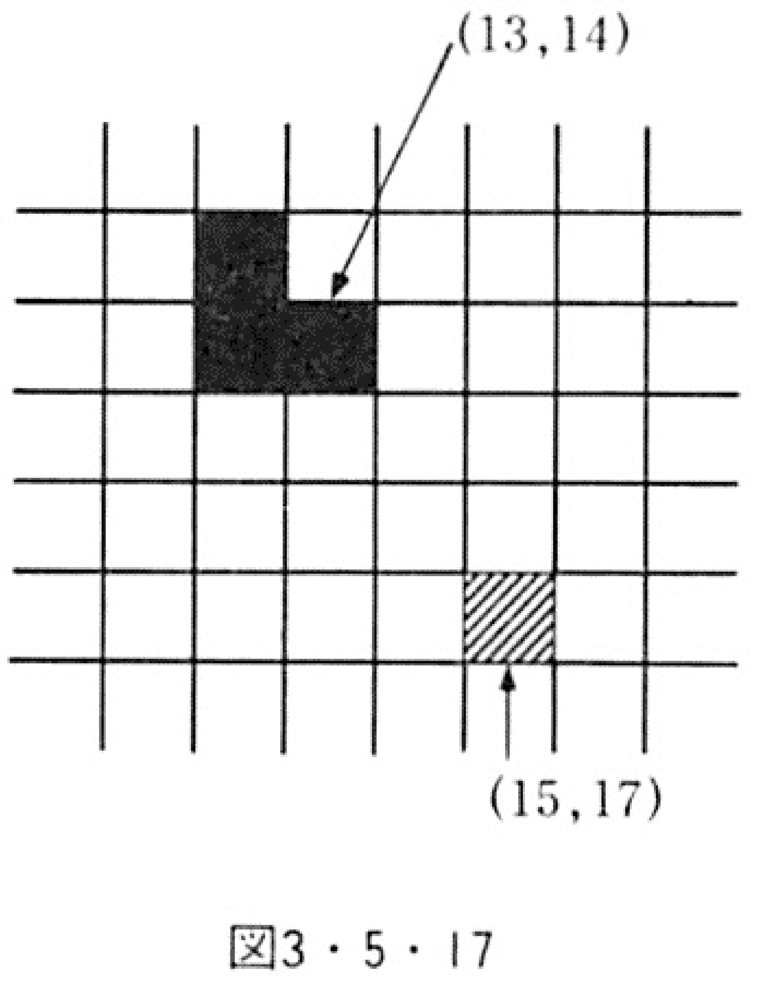


図3・5・17

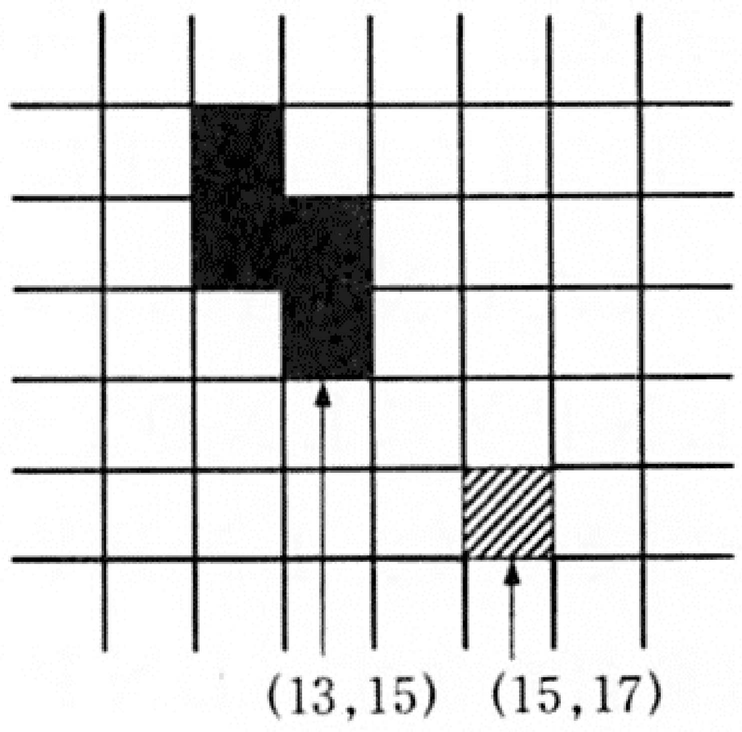


図3・5・18

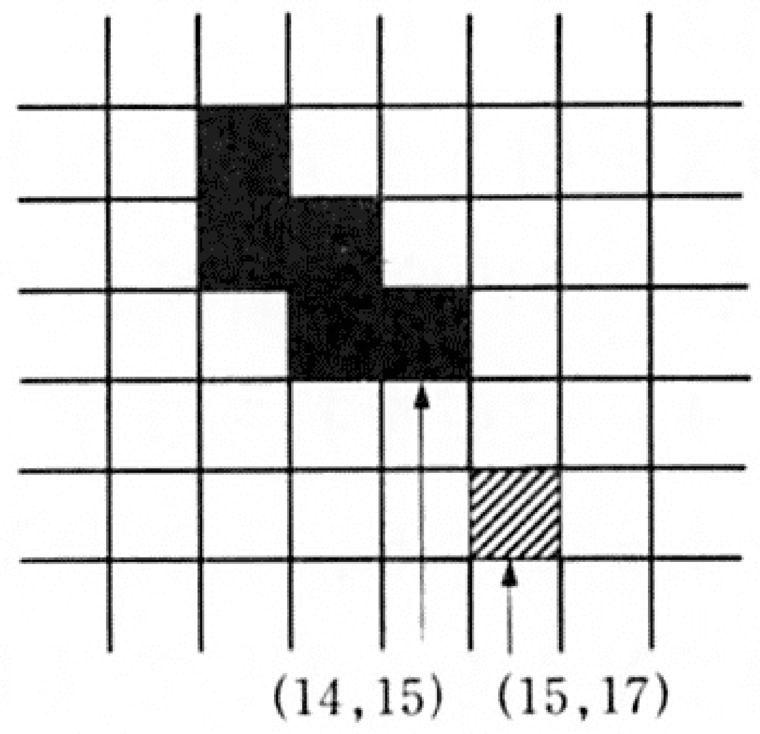


図3・5・19

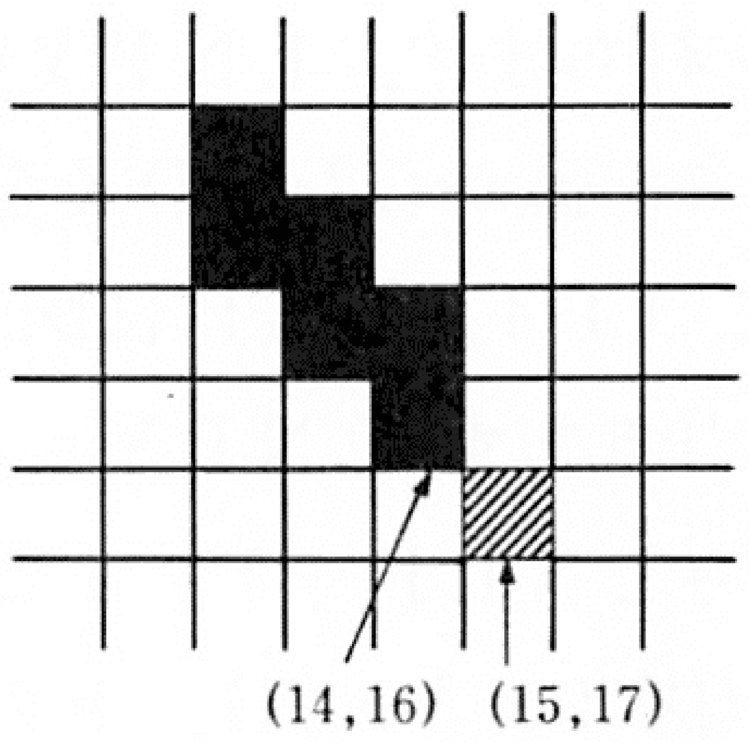


図3・5・20

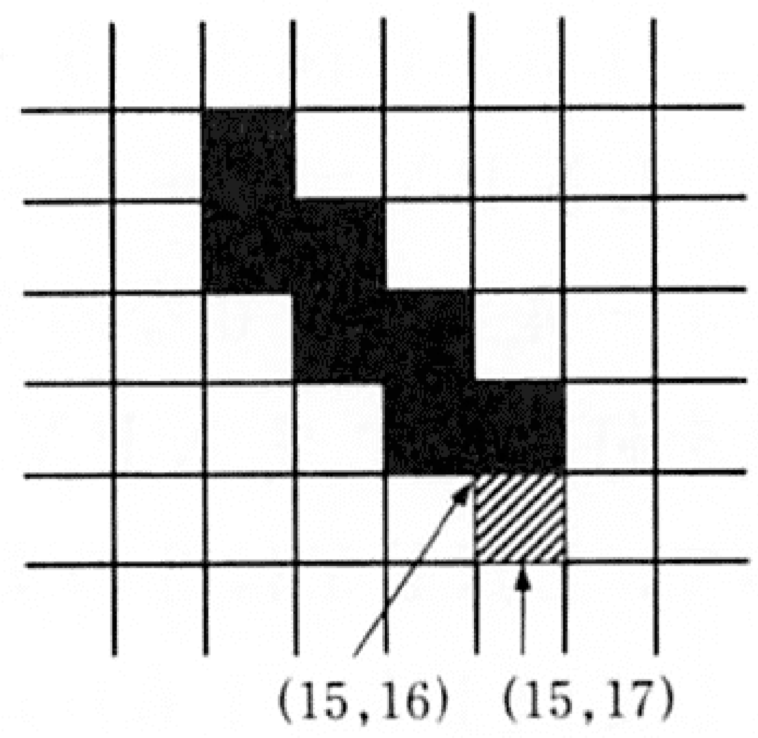


図3・5・21

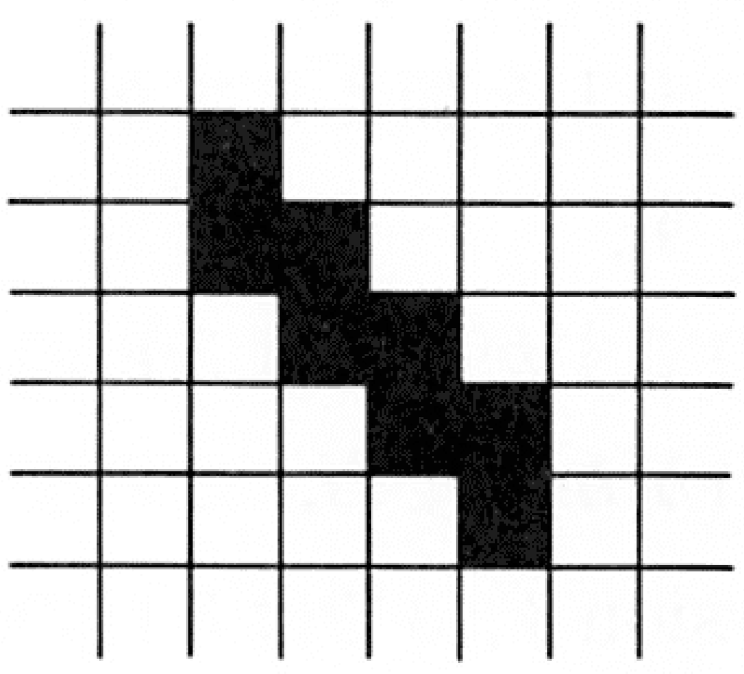


図3・5・22

```
110 '       LINE SIMULATOR
120 '
130 '
140 INPUT "X1=";X1
150 INPUT "Y1=";Y1
160 INPUT "X2=";X2
170 INPUT "Y2=";Y2
180 LNUX=SGN(X2-X1)
190 LNUY=SGN(Y2-Y1)
200 LNDX=ABS(X2-X1)+1
210 LNDY=ABS(Y2-Y1)+1
220 LNLP=LNDX
230 LNPY=0
240 PSET(X1,Y1)
250 LNPY=LNPY+LNDY
260 IF LNPY-LNDX<=0 GOTO310
270 LNPY=LNPY-LNDX
280 Y1=Y1+LNUY
290 PSET(X1,Y1)
300 GOTO 260
310 IF LNPY-LNDX<>0GOTO 350
320 LNPY=0
330 Y1=Y1+LNUY
340 PSET(X1,Y1)
350 X1=X1+LNUX
360 LNLP=LNLP-1
370 IF LNPY=0 THEN END
380 GOTO 240
```

3－5－2　CHAIN　　$16　($F0C4)

指定座標間を直線で結びます．座標点は３０個まで指定できます．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　　容	値
Ｄ３８３	ＣＬ	カラーコード	０～７
Ｄ３８４	Ｆ	ファンクションコード	０～４(５を除く)
Ｄ３８５	Ｎ	座標点数	２～３０
Ｄ３８６ ～ Ｄ３８７	Ｘ１	Ｘ１座標(１６ビット)	０～６３９ (グラフィック座標)
Ｄ３８８ ～ Ｄ３８９	Ｙ１	Ｙ１座標(１６ビット)	０～１９９ (グラフィック座標)
Ｄ３８Ａ ～ Ｄ３８Ｂ	Ｘ２	Ｘ２座標(１６ビット)	０～６３９ (グラフィック座標)
Ｄ３８Ｃ ～ Ｄ３８Ｄ	Ｙ２	Ｙ２座標(１６ビット)	０～１９９ (グラフィック座標)
⋮	⋮	⋮	⋮
Ｄ３８２＋ ４ｎ，４ｎ＋１	Ｘｎ	Ｘｎ座標（１６ビット)	０～６３９ (グラフィック座標)
Ｄ３８４＋ ４ｎ，４ｎ＋１	Ｙｎ	Ｙｎ座標（１６ビット)	０～１９９ (グラフィック座標)

主なサブルーチン

＄Ｅ４５２　　ＩＮＩＴＬＮ　　ラインイニシァライズ
＄Ｅ４４２　　ＧＡＸＣＨＫ　　グラフィック座標の範囲チェック
＄ＥＦ１Ａ　　ＬＩＮＥＳ　　ＬＩＮＥ実行

解　　説　このルーチンでは，以下のワークエリアを使用しています．

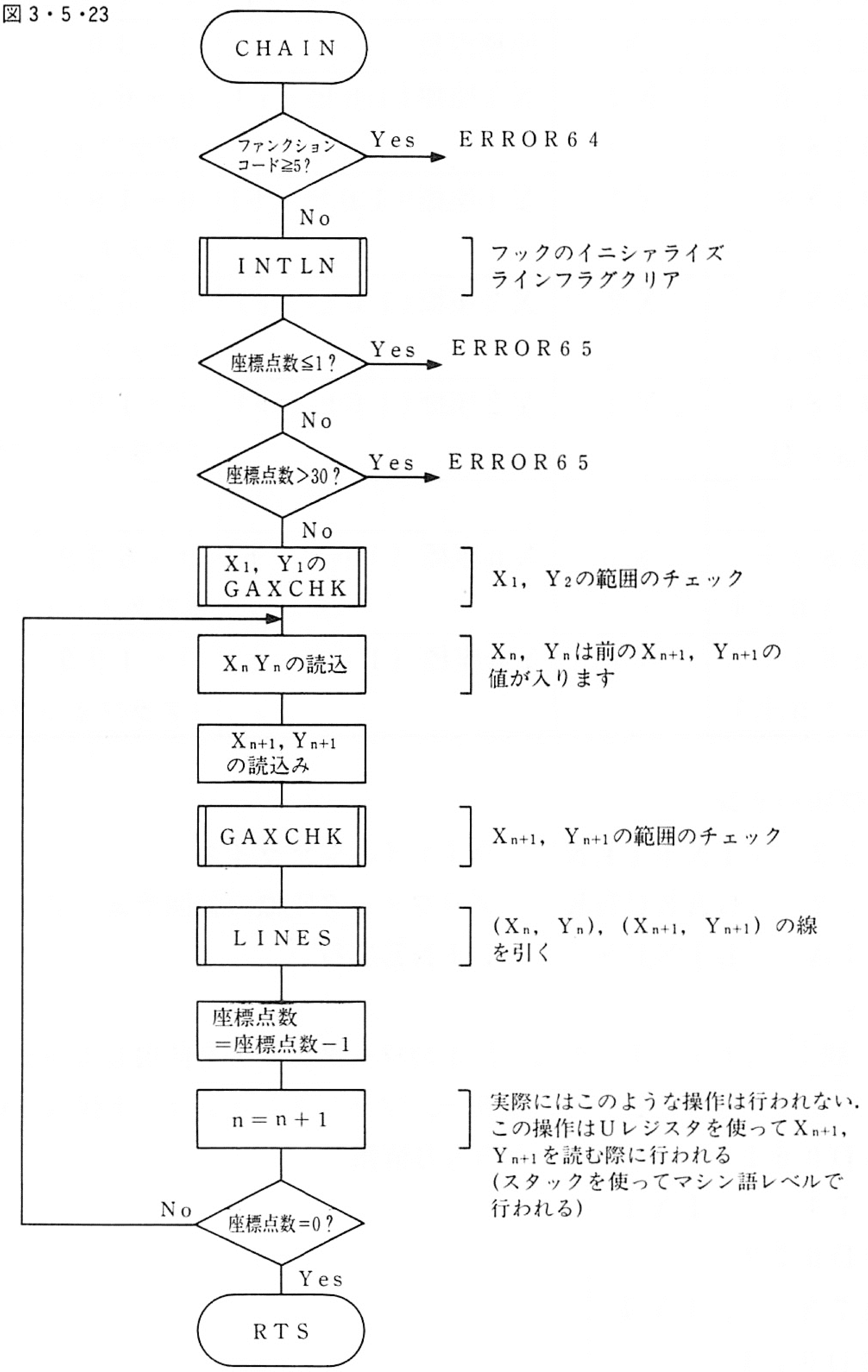
＄Ｄ０６０　．ＧＣＬ　　カラー，ファンクションコード代入，ＧＰＤＳＰ
　～＄Ｄ０６１　　　　Ｉの入力情報
＄Ｄ０７１　．ＬＸ１
　～＄Ｄ０７２
＄Ｄ０７３　．ＬＹ１
　～＄Ｄ０７４

$D075　　.LX2
　~$D076
$D077　　.LY2
　~$D078

これらはすべてLINESの入力情報として使われます.

このルーチンは比較的短かくて60バイトたらずで構成されています. その理由は, LINE処理系のLINESルーチンを下位ルーチンとして, 効率よく呼び出しているからです.

図3・5・23



3－5－3　POINT　　$17　($F412)

指定された点を表示します．点の位置は２０個まで指定できます．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　　容	値
Ｄ３８３	Ｎ	表示点数	１～２０
Ｄ３８４ ～ Ｄ３８５	Ｘ１	Ｘ１座標（１６ビット）	０～６３９ （グラフィック座標）
Ｄ３８６ ～ Ｄ３８７	Ｙ１	Ｙ１座標（１６ビット）	０～１９９ （グラフィック座標）
Ｄ３８８	ＣＬ１	カラーコード	０～７
Ｄ３８９	Ｆ１	ファンクションコード	０～４（５を除く）
⋮	⋮	⋮	⋮
D384＋6(n－1) ，＋1	Ｘｎ	Ｘｎ座標（１６ビット）	０～６３９ （グラフィック座標）
D386＋6(n－1) ，＋1	Ｙｎ	Ｙｎ座標（１６ビット）	０～１９９ （グラフィック座標）
D388＋(6n－1)	ＣＬｎ	カラーコード	０～７
D389＋(6n－1)	Ｆｎ	ファンクションコード	０～４（５を除く）

主なサブルーチン

$Ｅ４４２	ＧＡＸＣＨＫ	グラフィック座標の範囲チェック
$Ｅ４６９	ＧＰＤＳＰＩ	フックのイニシァライズ
$Ｅ４８Ｆ	ＧＰＤＳＰ	指定された点にファンクションコード，カラーに合わせて点を打つ

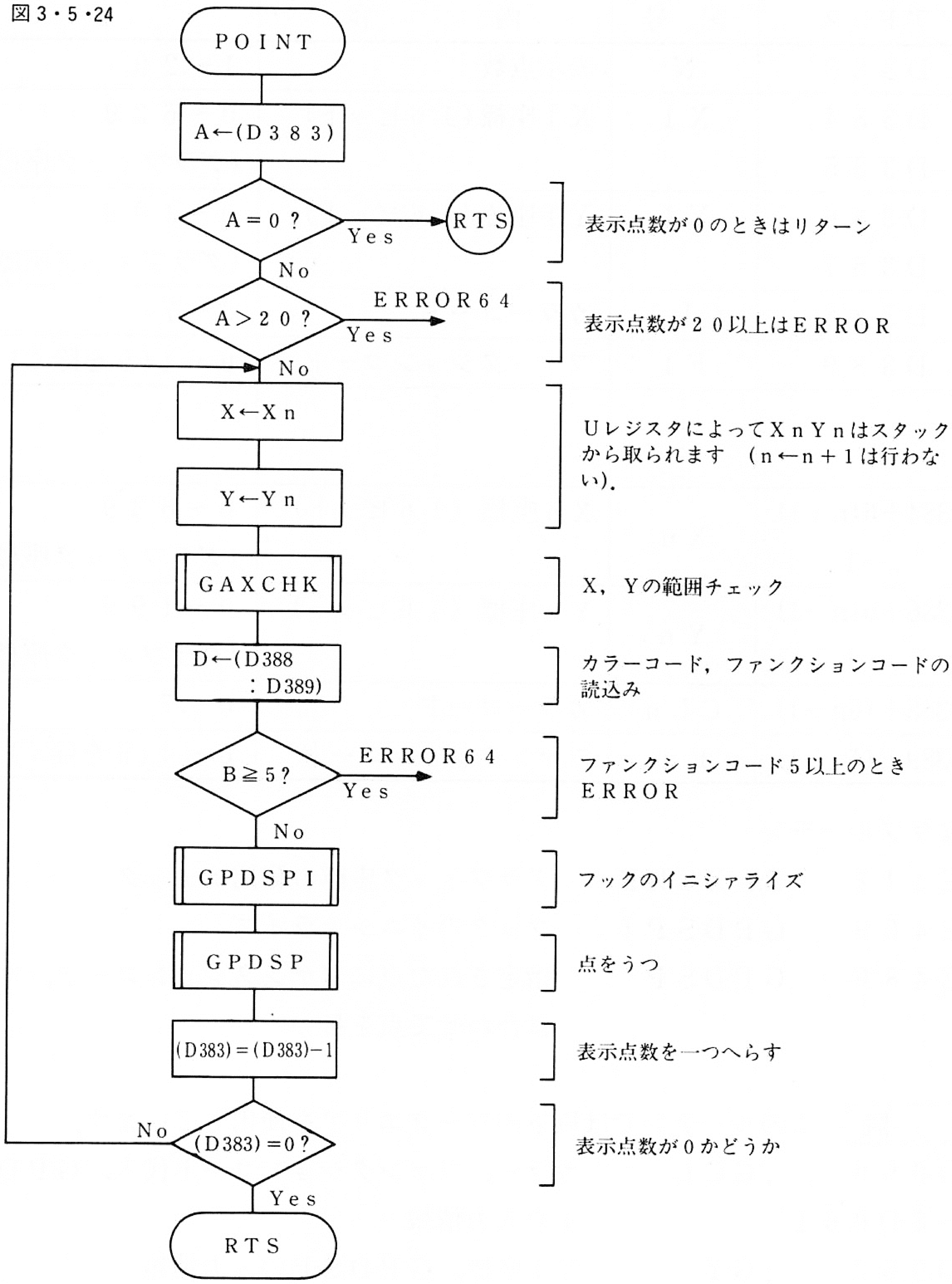
解　説　このルーチンでは以下のワークエリアを使用しています．

$Ｄ０６０ ～$Ｄ０６１	.ＧＣＬ	カラー，ファンクションコード代入，ＧＰＤＳＰＩの入力情報
$Ｄ０６２ ～$Ｄ０６３	.ＧＹ	Ｙ１座標，ＧＰＤＳＰの入力情報
$Ｄ０６４ ～$Ｄ０６５	.ＧＸ	Ｘ１座標，ＧＰＤＳＰの入力情報

ＰＯＩＮＴ処理系ルーチン自体は非常に短かくて５０バイトぐらいしかありません．それは，手足となるグラフィック汎用サブルーチン群を，効率よく呼び出して使用しているからです．

以下にこのルーチンのフローチャートを示します（図３・５・24　参照）．

図３・５・24



※このフローチャートは完全にマシン語レベルではなく不必要なところは省略してあります

３－５－４　ＰＡＩＮＴ　＄１８　（＄ＦＡ８Ａ）

指定された座標点から，指定された境界色で囲まれた部分を塗りつぶします．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
Ｄ３８３ ～ Ｄ３８４	Ｘ	Ｘ座標	０～６３９ (グラフィック座標)
Ｄ３８５ ～ Ｄ３８６	Ｙ	Ｙ座標	０～１９９ (グラフィック座標)
Ｄ３８７	Ｐｃ	ペイントカラーコード	０～７
Ｄ３８８	Ｎｃ	境界色の数	１～８
Ｄ３８９	Bc_1	境界色カラーコード$_1$	０～７
⋮	⋮	⋮	⋮
D388＋Nc	Bc_n	境界色カラーコード$_n$	０～７

主なサブルーチン

＄Ｆ２ＥＤ	ＣＬＥＸＴ	カラーの初期設定
＄Ｅ４４２	ＧＡＸＣＨＫ	グラフィック座標の範囲のチェック
＄Ｅ４５２	ＩＮＩＴＬＮ	ラインイニシァライズ
＄ＦＣＣ２	ＰＡＳＴＰ１	右の境界色をサーチする
＄ＦＣＣ７	ＰＡＳＴＰ２	左の境界色をサーチする
＄ＦＣＣＣ	ＰＡＳＴＰ３	右の境界色でないところをサーチする
＄ＦＣＤＤ	ＰＡＳＴＰ４	左の境界色でないところをサーチする

補助ワークエリア（＄Ｄ０Ａ０～＄Ｄ０Ｂ８はＬＩＮＥ，ＧＢＬＫと共通）

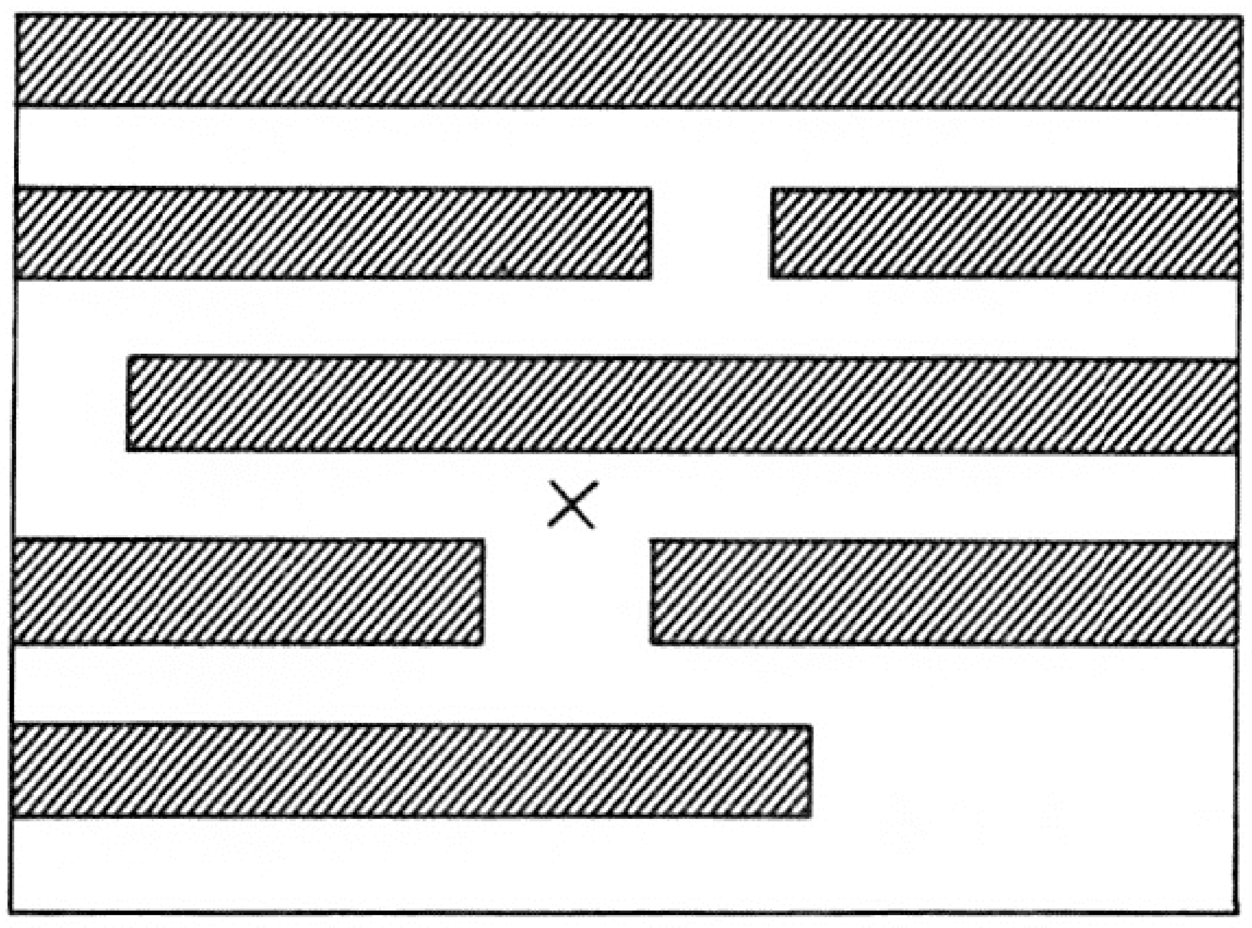
＄Ｄ０Ｂ９ ～＄Ｄ０ＢＡ	ＰＡＳＡＶＥ	Ｘ座標の保存
＄Ｄ０ＢＢ	ＰＡＢＲＡＮ	スタック復帰後のジャンプアドレスの指定
＄Ｄ０ＢＤ ～＄Ｄ０Ｃ４	ＰＡＪＯＨＯ	ペイント座標処理用ワークエリア
＄Ｄ０Ｃ４ ～＄Ｄ０Ｃ５	ＰＡＤＩＲＹ	Ｙ座標の保存

＄D0C6　　PAYADD　Y座標＊80の保存
　～＄D0C7
＄D0C8　　PANXIY　Y座標＊80の保存
　～＄D0C9
＄D0CA　　PACURY　以前のY座標＊80の保存
　～＄D0CB
＄D0CD　　PASTKP　Uレジスタの保存（スタックポインタ）
＄D0CF　　PASTAK　ペイントルーチン用スタック
　～＄D2BF

解説　このルーチンは，再帰呼出しという方法を利用しています．このルーチンを簡単に説明します．図3・3・25を参照して下さい．PAINTというのは，このように迷路のすべての道を歩いてゆくと考えて下さい（図3・5・26）．このたどり方なのですが，そのたどり方は次の3つに分かれます．①ひとつは左側にはそのY座標上では行けなくてY座標を±1すれば行けるとき，②もうひとつは右側にはそのY座標上では行けなくてY座標を±1すれば行けるとき，③あとひとつは道の途中が分かれているときです（図3・5・27①，②，③参照）．

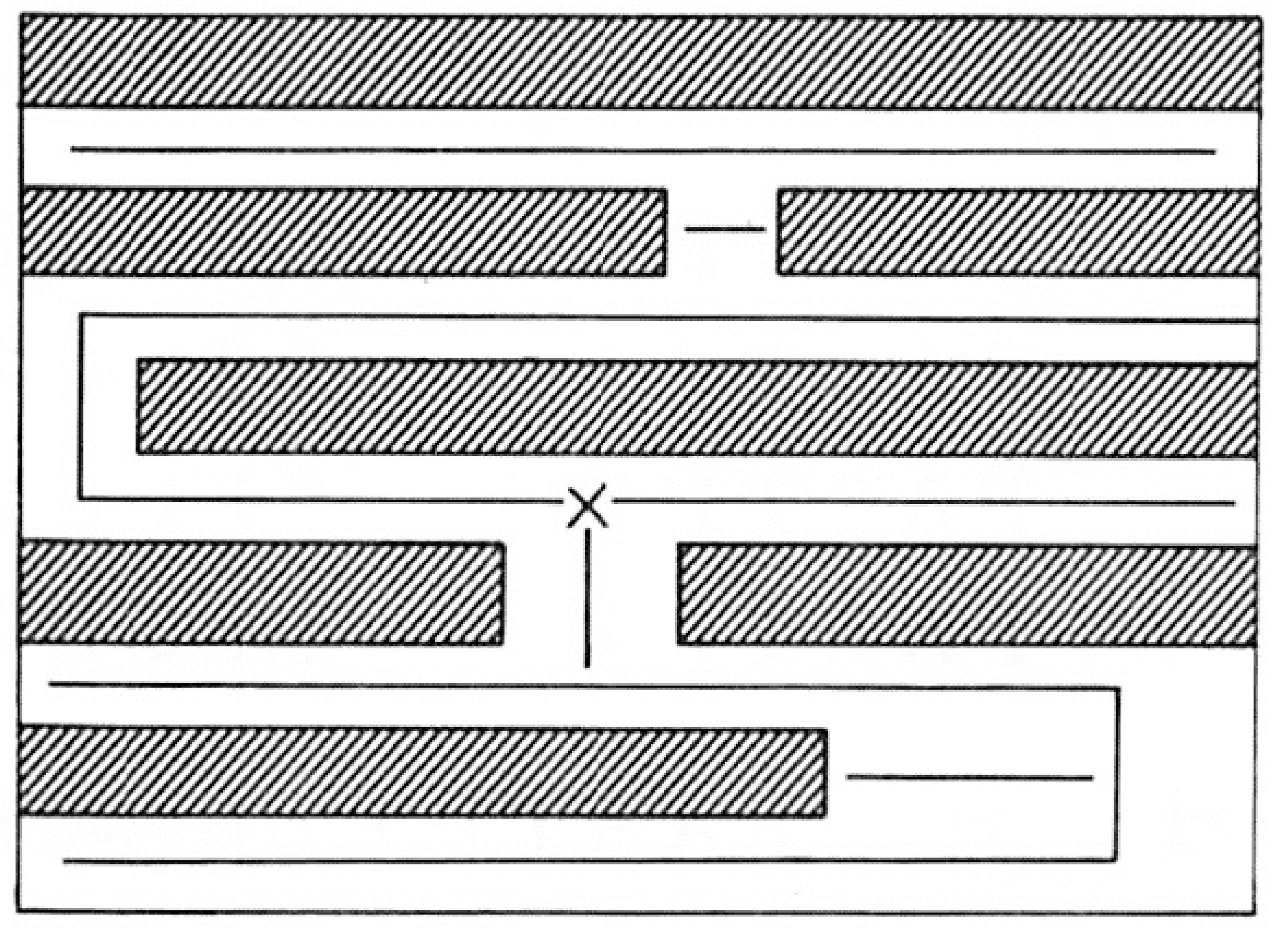
たどっている最中にこのようなところに出くわしたら，次のことをしなければなりません．①では，上か下へ行けるときは，現在の状態と上か下へ今度戻ってきたときどちらに行けばよいかという情報をスタックに保存します．これをくり返し，もし行き止まりだったら，そのスタックに入れたところへ戻ります．このとき行く方向が上か下かは決定されているのでその方向へ進みます．この操作をスタックがなくなるまでくり返すと左側は完全にたどることができます．右についても同様のことをします．③については，同じようにこの分かれ道があったらここに戻ってきたとき上へ行くか下へ行くかの情報と現在の状態をスタックしてこの道に入るのですが，この場合は①と②の操作が完了した時点でもとに戻ります．①と②では，上へ行くか下へ行くかは決まっています．上へ行ったら次の分かれ道では下へ，その次は上へと交互にしなければなりません．

実際には①も②も③も独立ではなくて複雑にからみあっているのですが，簡単にするためこのアルゴリズムを①だけについて書くと，図3・5・28のようになります．



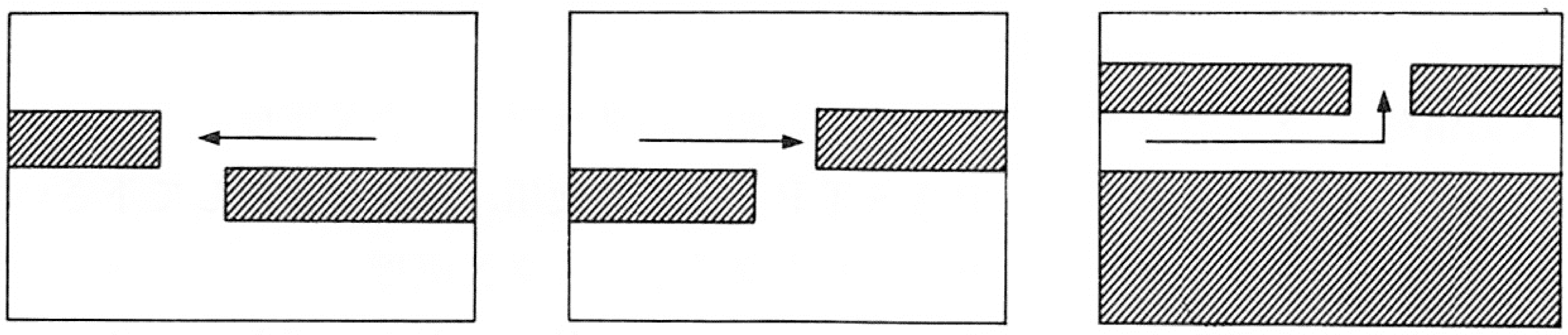
× 指定された点 ▨ 境界色

図3・5・25



たどったところを塗るとＰＡＩＮＴになる

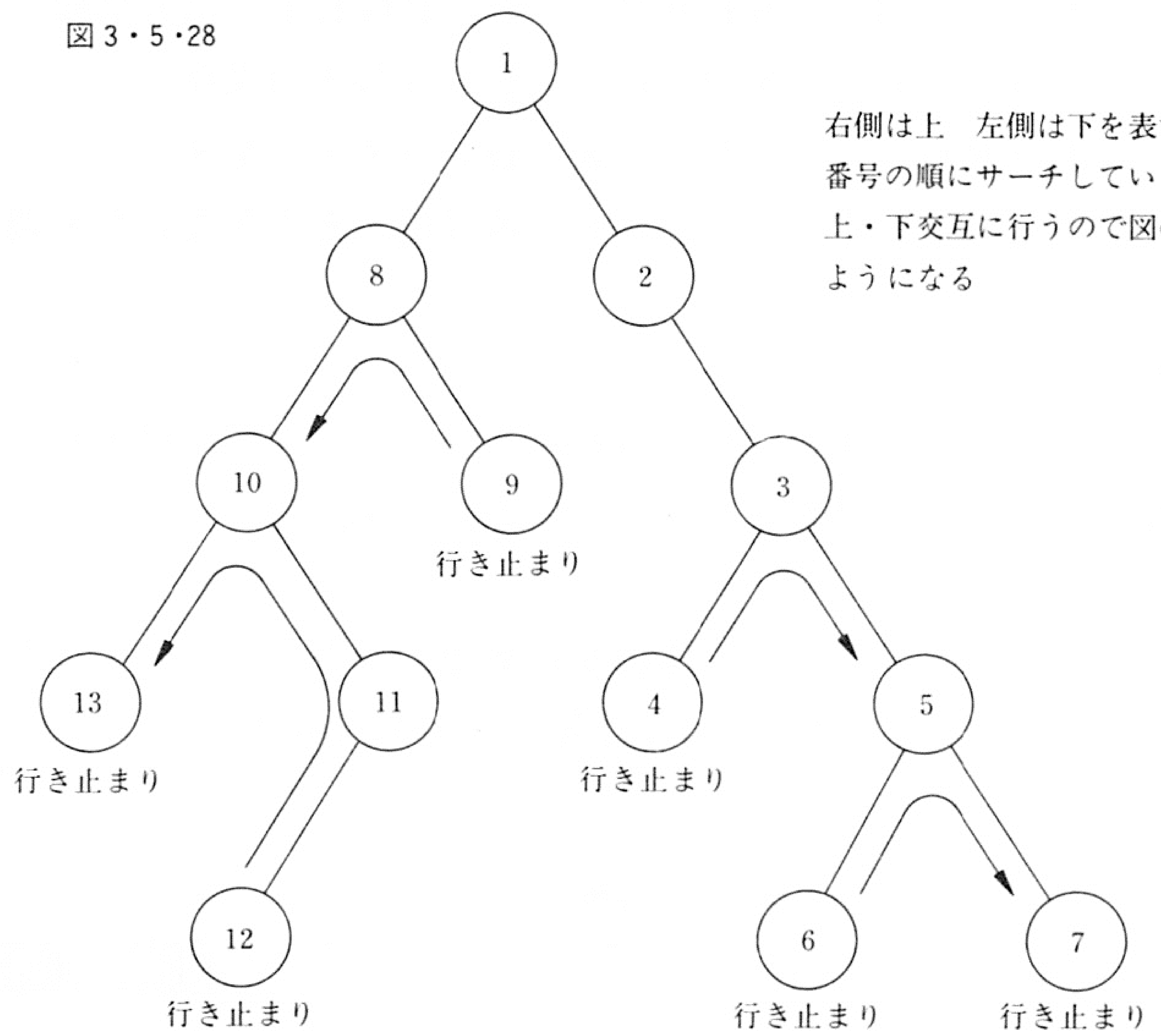
図3・5・26



① ② ③

図3・5・27

図３・５・28



右側は上　左側は下を表す．
番号の順にサーチしていく．
上・下交互に行うので図の
ようになる

下へおりるのは左側をその点より先までさがすことです

PASTP1, PASTP2, PASTP3, PASTP4

PASTP1	$FCC2～$FD5B
PASTP2	$FCC7
PASTP3	$FCCC
PASTP4	$FCDD

機能	PASTP1	右の境界色をサーチする	
	PASTP2	左の境界色をサーチする	
	PASTP3	右の境界色でないところをサーチする	
	PASTP4	左の境界色でないところをサーチする	
レジスタ	A, B, X, U		
入力情報	Xレジスタ	サーチする始点のグラフィックX座標	
	Dレジスタ	PASTP3, 4で使用, 解説を参照して下さい.	
出力情報	Xレジスタ	サーチしたグラフィックX座標	
ワークエリア	PARL	$D0B2～$D0B3	PASTP1, 2, 3, 4のどれであるかを保存
	PALIMT	$D0BC	X座標の限度を代入
	PASIMG	$D0B4	PABGETの出力情報
	PAXOFF	$D0CC	Bレジスタの保存
サブルーチン	LSRD3	$E3F8	Dレジスタ ￥8
	PABGET	$FC80	境界色であるかないかを調べる

解説

PASTP1　このルーチンは指定されたグラフィックX座標より右側で一番左側にある境界色の点の座標をXレジスタに返します. 図3・5・29を参照して下さい.

ただし指定されたグラフィック座標が境界色の場合は, その座標がXレジスタに返されます. また, 境界がない場合はXレジスタの値は$0280となります.

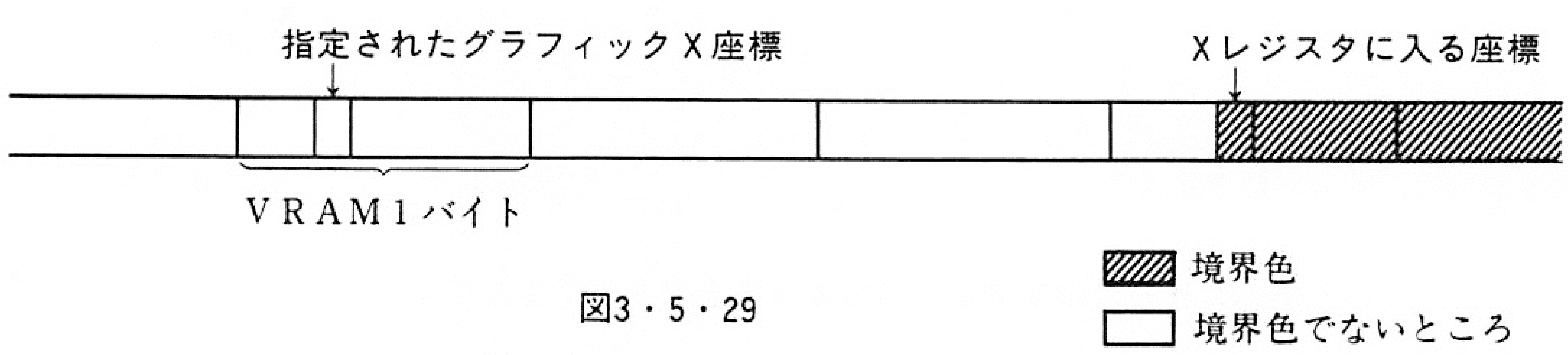


図3・5・29

ＰＡＳＴＰ２　このルーチンは指定されたグラフィックＸ座標より左側で一番右側にある境界色の座標をＸレジスタに返します．図３・５・30を参照して下さい．この場合も指定されたグラフィック座標が境界色の場合は，その座標がＸレジスタに返されます．ただし境界がないときはＸレジスタの値は＄ＦＦＦＦとなります．

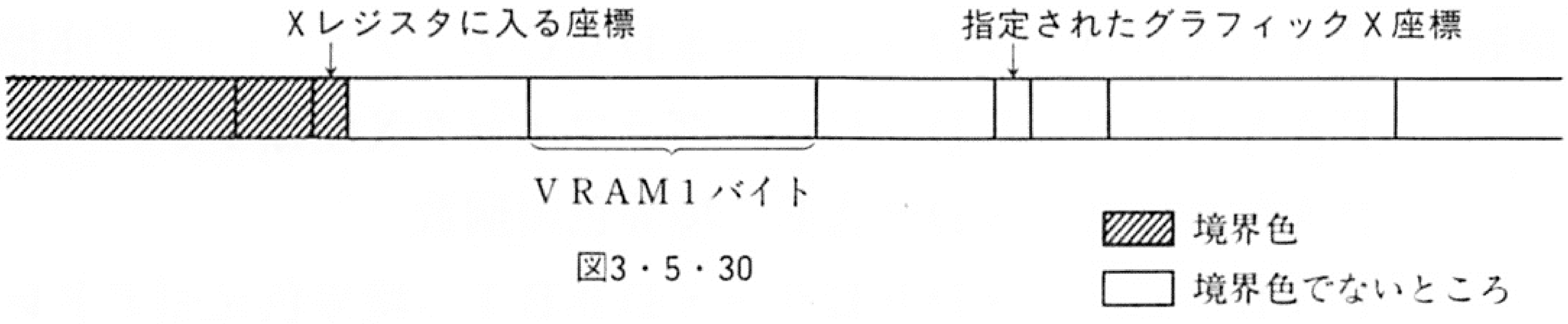


図3・5・30

ＰＡＳＴＰ３　このルーチンは指定されたグラフィックＸ座標より右側で一番左側にある境界色でない点の座標をＸレジスタに返します．しかしこのルーチンではＤレジスタの示すＸ座標によってこのサーチが制限されます．図３・５・31を参照して下さい．つまりこのＤレジスタのＸ座標のあるバイトより，境界色でないところが右側にあるときは，Ｘレジスタの値が＄０２８０となります．

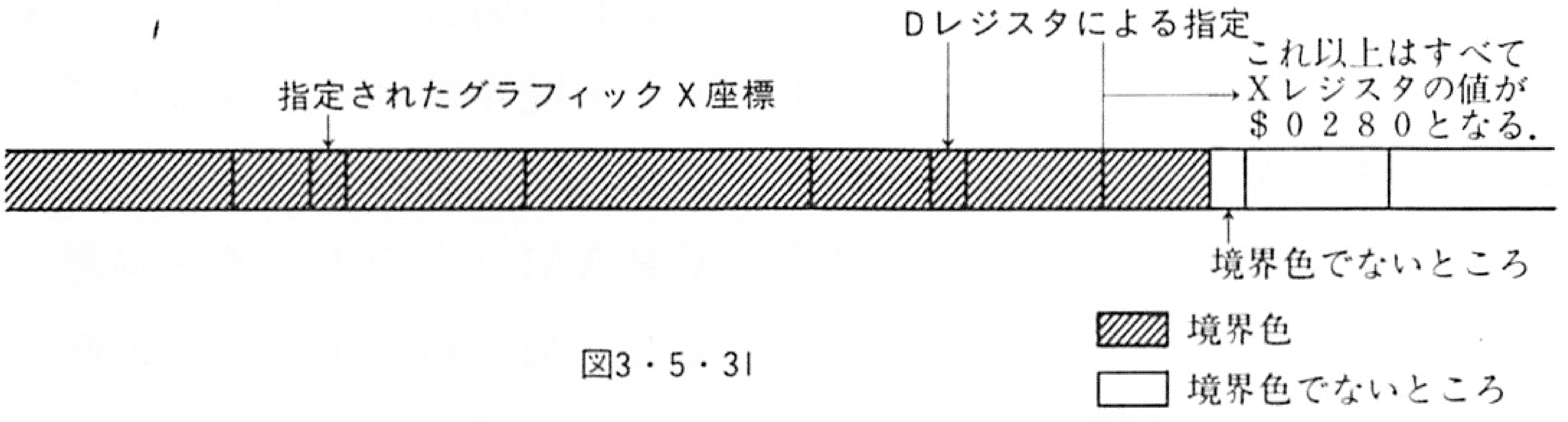


図3・5・31

ＰＡＳＴＰ４　このルーチンは指定されたグラフィックＸ座標より左側で一番右側にある境界色でない点の座標をＸレジスタに返します．このルーチンもＤレジスタによって制限されます．このＤレジスタのＸ座標のあるバイトより，境界色でないところが左側にあるときは，Ｘレジスタの値が＄ＦＦＦＦとなります．

ＦＭ－８　Version ３．３ではこのようなＤレジスタによる制御を行っていません．

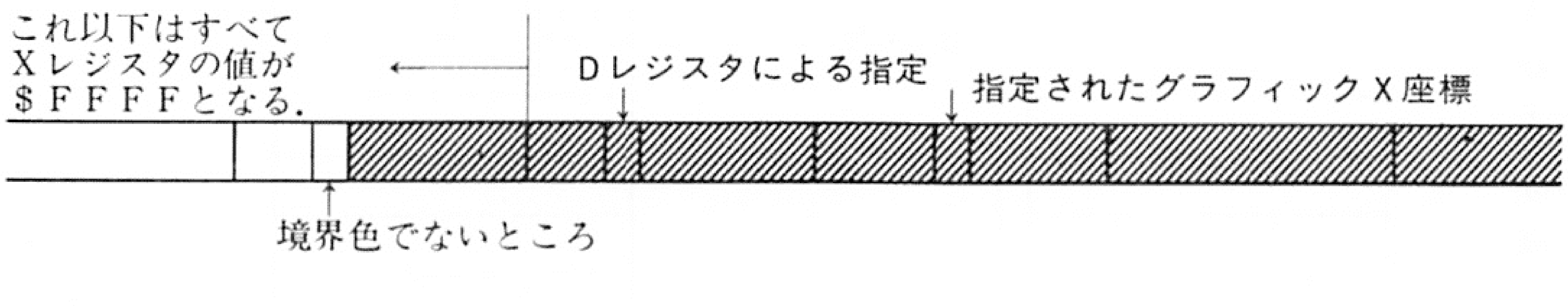


図3・5・32

ＰＡＢＧＥＴ

ＰＡＢＧＥＴ　＄ＦＣ８０～＄ＦＣＣ１
ＧＢＬＫＤＴ　＄ＦＣ８６

機能　境界色であるかないかを調べます．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　ＰＡＹＡＤＤ　＄Ｄ０Ｃ６～＄Ｄ０Ｃ７　グラフィックＹ座標×＄50
ＰＡＸＯＦＦ　＄Ｄ０ＣＣ　グラフィックＸ座標￥８
ＰＡＮＢＣＬ　＄Ｄ０ＡＦ　境界色の個数
ＰＡＢＣＬＥ　＄Ｄ０Ｂ０～＄Ｄ０Ｂ１　境界色先頭アドレス．境界色の代入の仕方は解説参照
ＰＡＳＣＮＤ　＄Ｄ０Ｂ３　＄００　境界色
＄ＦＦ　境界色でない

出力情報　ＰＡＳＩＭＧ　＄Ｄ０Ｂ４　＄Ｄ０Ｂ３＝＄００の場合：指定されたグラフィックＸ，Ｙ座標を含む１バイトを境界色は1，他は0にして代入
＄Ｄ０Ｂ３＝＄ＦＦの場合：グラフィックＸ，Ｙ座標を含む１バイトを境界色は0，他は１にして代入

ワークエリア　ＰＡＶＢ　＄Ｄ０Ｂ６　ＶＲＡＭ，青の１バイト退避
ＰＡＶＲ　＄Ｄ０Ｂ７　ＶＲＡＭ，赤の１バイト退避
ＰＡＶＧ　＄Ｄ０Ｂ８　ＶＲＡＭ，緑の１バイト退避

解説　このルーチンはＰＡＳＴＰルーチンで境界色や境界色でないところをさがすのに利用されます．境界色は図３・５・33のように３バイトで１つの色に対応し，その色を含んでいる３原色に対応するところをすべて１でうめます．この先頭アドレスは＄Ｄ０Ｂ０～＄Ｄ０Ｂ１で示されます．図３・５・33は赤が境界色となっている例です．

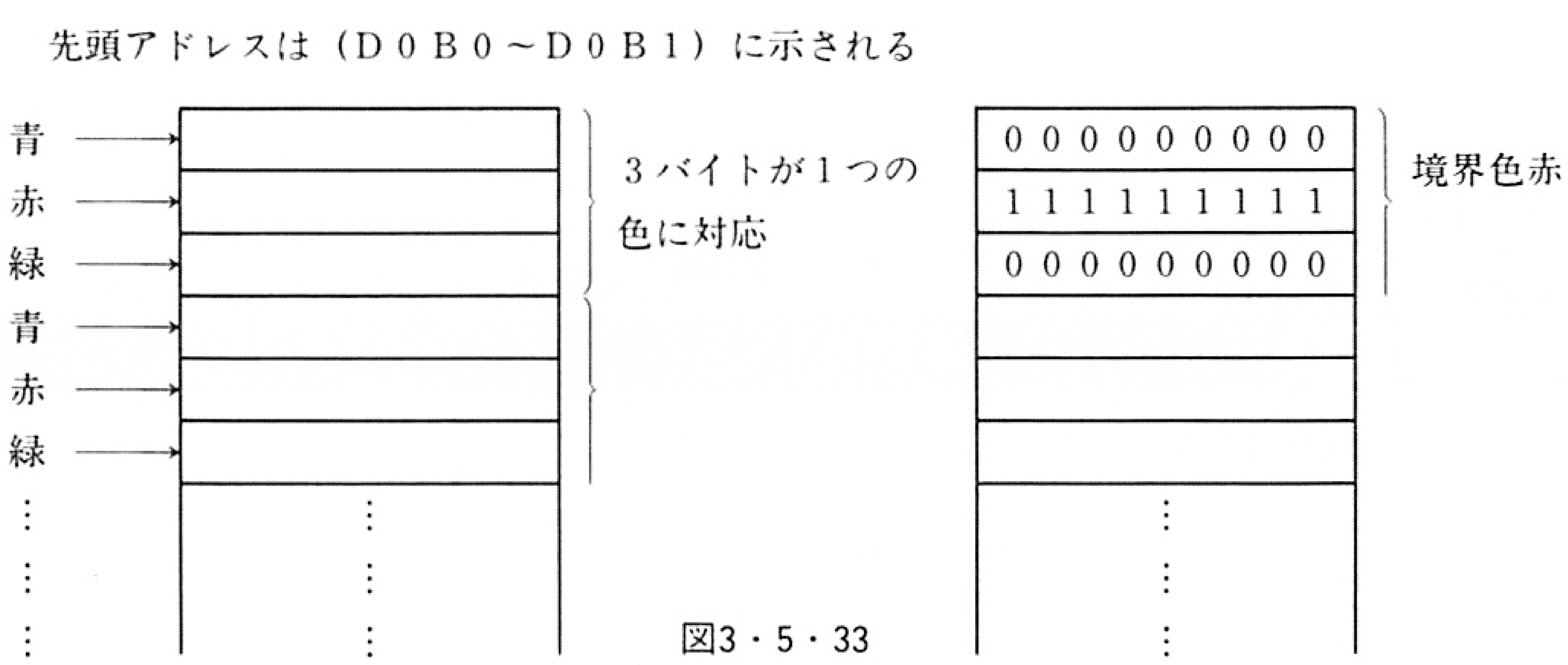


図3・5・33

このルーチンでどのようにＰＡＳＩＭＧがセットされるか図３・５・34を使って説明します．

例えば境界色を赤，青と決定し，グラフィック座標Ｘ，Ｙを含むＶＲＡＭ１バイトのビットパターンが①のようになっているとＰＡＳＩＭＧはＰＡＳＣＮＤの値によって②のように決定されます．

このルーチンを利用することによって，ＶＲＡＭの好きな色だけ取り出してすべてを消してしまうことなども可能です．

なお，このルーチンはGet Block １処理系からも呼び出されており，そのためのエントリポイントＧＢＬＫＤＴが用意されております（３－６－３節　参照）．

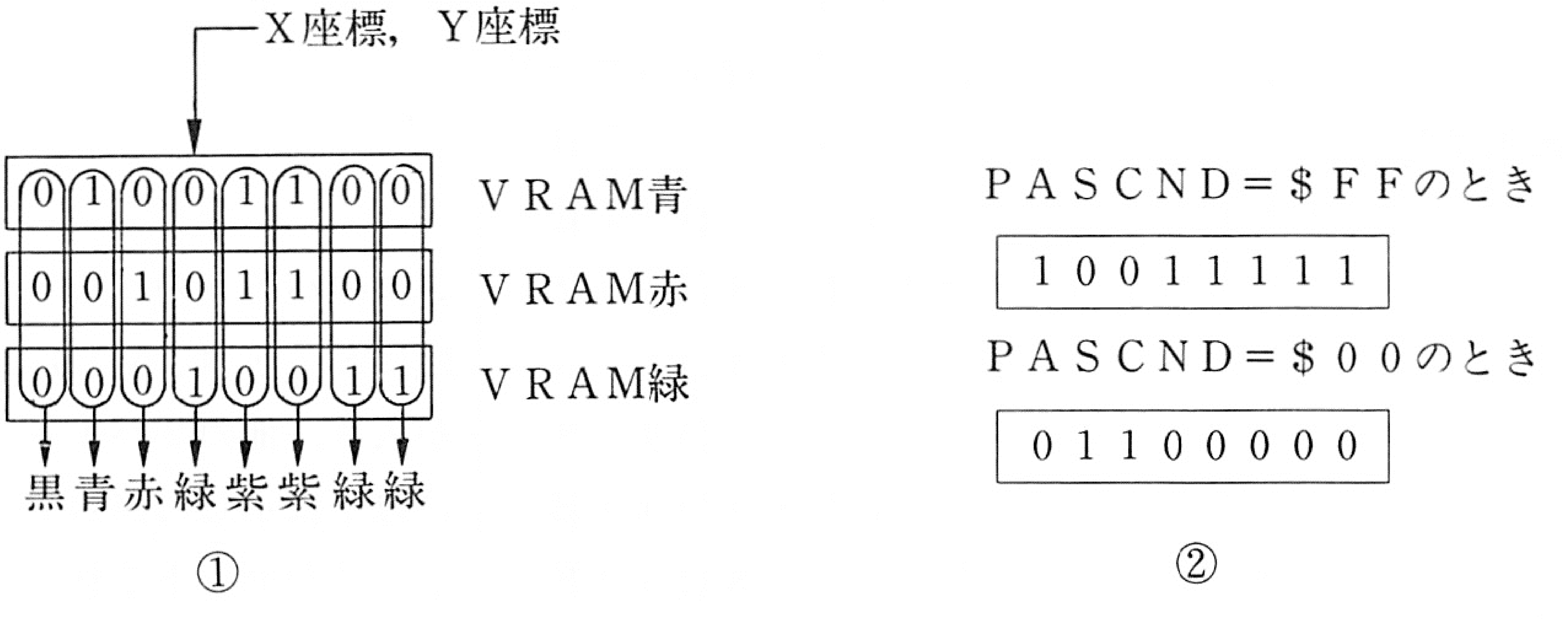


図３・５・34

3－5－5　グラフィック汎用サブルーチン

1）ＩＮＴＬＮ

＄Ｅ４５２　～　＄Ｅ４８Ｄ

機　能　ファンクションコードとカラーコードを調べそれにあったフックを初期設定します．

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　．ＧＣＬ　＄Ｄ０６０～＄Ｄ０６１　＄Ｄ０６０：カラーコード，＄Ｄ０６１：ファンクションコード

．．ＢＣＬ　＄Ｄ０２Ｆ　背景色コード

出力情報

ＢＬＵＳＩＤ	＄Ｄ０７９～＄Ｄ０７Ｂ	３つのバイトのうち先頭に＄７Ｅ（ＪＭＰ）が入り，他の２バイトに以下に示した図のアドレスがセットされる
ＲＥＤＳＩＤ	＄Ｄ０７Ｃ～＄Ｄ０７Ｅ	
ＧＲＥＳＩＤ	＄Ｄ０７Ｆ～＄Ｄ０８２	
ＢＬＵＤＳＰ	＄Ｄ０８３～＄Ｄ０８５	
ＲＥＤＤＳＰ	＄Ｄ０８６～＄Ｄ０８８	
ＧＲＥＤＳＰ	＄Ｄ０８９～＄Ｄ０８Ｂ	
ＢＬＵＬＩＮ	＄Ｄ０８Ｃ～＄Ｄ０８Ｅ	
ＲＥＤＬＩＮ	＄Ｄ０８Ｆ～＄Ｄ０９１	
ＧＲＥＬＩＮ	＄Ｄ０９２～＄Ｄ０９４	

ＬＮＣＨＲ　＄Ｄ０Ａ０＝０に設定

解　説　このルーチンでは，まずキャラクタラインであるかどうかを調べるフラグ（＄Ｄ０Ａ０）を０に設定します（キャラクタラインでないことを示す）．

次に，＄Ｄ０７９～＄Ｄ０９４までに設定されたフックを出力情報で示したように設定します(図３・５・35，図３・５・36)．この図は，２つの特別な場合についてあげてあります．一つはＲ＝０，Ｇ＝０，Ｂ＝０（黒）でもう一つは　Ｒ＝１，Ｇ＝１，Ｂ＝１（白）のときです．しかしラベル名のＢＬＵＳＩＤ，ＢＬＵＤＳＰ，ＢＬＵＬＩＮは，Ｂが１か０かで決まります．他の，はじめにＲＥＤのつくラベルは，Ｒが１か０によって，ＧＲＥのつくラベルは，Ｇが１か０かによって決定されます．ですからこの２つの図だけですべてのＲＧＢの組合わせができます．例えばＲ＝１，Ｇ＝０，Ｂ＝１（水色）では，ＢＬＵ――は，図３・５・36のように決まり，ＲＥＤ――は，図３・５・36のようになり，ＧＲＥ――は，図３・５・35のようになります（ここで“――”はＳＩＤ，ＤＳＰ，ＬＩＮのい

ずれかを表す).

図3・5・35　RGBの指定がR＝0　G＝0　B＝0のとき

ラベル名	SET	PRESET	OR	AND	EOR
BLUSID	F647	F647	F657	F647	F657
REDSID	F647	F647	F657	F647	F657
GRESID	F647	F647	F657	F647	F657
BLUDSP	E4CD	E4CD	E4CC	E4CD	E4CC
REDDSP	E4CD	E4CD	E4CC	E4CD	E4CC
GREDSP	E4CD	E4CD	E4CC	E4CD	E4CC
BLULIN	F092	F092	F0A2	F092	F0A2
REDLIN	F092	F092	F0A2	F092	F0A2
GRELIN	F092	F092	F0A2	F092	F0A2

図3・5・36　RGBの指定が　R＝1　G＝1　B＝1　のとき

ラベル名	SET	PRESET	OR	AND	EOR
BLUSID	F631	F631	F631	F657	F65E
REDSID	F631	F631	F631	F657	F65E
GRESID	F631	F631	F631	F657	F65E
BLUDSP	F094	F094	F094	F0A2	F0A3
REDDSP	F094	F094	F094	F0A2	F0A3
GREDSP	F094	F094	F094	F0A2	F0A3
BLULIN	E4C1	E4C1	E4C1	E4CC	E4DA
REDLIN	E4C1	E4C1	E4C1	E4CC	E4DA
GRELIN	E4C1	E4C1	E4C1	E4CC	E4DA

ここで気をつけてほしいのは，SET，OR，AND，EORは，カラーコードによってこのフックを設定するのですが，PRESETだけは背景色によって設定するということです.

この図にあげてあるアドレスは，VRAM上ドットを打つためのルーチンのエントリアドレスです. このルーチンについては次の項を参照して下さい.

2）ＶＲＡＭ記入

ｉ）ＧＰＯＲ，ＧＰＮＯＰ，ＧＰＡＮＤ，ＧＰＥＯＲ

ＧＰＯＲ	＄Ｅ４Ｃ１	～　＄Ｅ４ＣＣ
ＧＰＮＯＰ	＄Ｅ４ＣＣ	
ＧＰＡＮＤ	＄Ｅ４ＣＤ	～　＄Ｅ４Ｄ９
ＧＰＥＯＲ	＄Ｅ４ＤＡ	～　＄Ｅ４Ｅ５

機　能　Ｕレジスタの示す番地のＶＲＡＭとＢレジスタのＯＲ，ＡＮＤ，ＥＯＲをとり，Ｕレジスタの示すＶＲＡＭに書き込みます．

レジスタ　Ａ，Ｂ，Ｕ

入力情報　Ｕレジスタ　書き込むＶＲＡＭのアドレス
　　　　　Ｂレジスタ　書き込むバイトのビットパターン

ワークエリア　ＶＡＣＣ　＄Ｄ４０９

解　説　このルーチン群は以下の機能を示します．

ＧＰＯＲ　…Ｕレジスタの示す番地とＢレジスタのＯＲをとります．
ＧＰＮＯＰ…何もしません．
ＧＰＡＮＤ…Ｕレジスタの示す番地とＢレジスタのＡＮＤをとります．
ＧＰＥＯＲ…Ｕレジスタの示す番地とＢレジスタのＥＯＲをとります．

ii）ＬＮＡＮＤ，ＬＮＯＲ，ＬＮＮＯＰ，ＬＮＥＯＲ

ＬＮＡＮＤ	＄Ｆ０９２～＄Ｅ０Ａ２
ＬＮＯＲ	＄Ｆ０９４～＄Ｆ０Ａ２
ＬＮＮＯＰ	＄Ｆ０Ａ２
ＬＮＥＯＲ	＄Ｆ０Ａ３～＄Ｆ０ＡＦ

機　能　Ｕレジスタの示す番地のＶＲＡＭからＢレジスタ分だけ進んだところまでのＶＲＡＭを書き込みます．

レジスタ　Ａ，Ｂ，Ｕ

入力情報　Ｕレジスタ　書き込むＶＲＡＭの先頭アドレス
　　　　　Ｂレジスタ　書き込むバイト数＋１

解　説　これらのルーチン群は以下の機能を示します．

ＬＮＡＮＤ…Ｕレジスタの示す番地からＵ＋Ｂの示す番地までを＄００でうめていきます．つまり０でＬＩＮＥを描きます．
ＬＮＯＲ　…Ｕレジスタの示す番地からＵ＋Ｂの示す番地までを＄ＦＦでうめていきます．つまり１でＬＩＮＥを描きます．

ＬＮＮＯＰ…何もしません．

ＬＮＥＯＲ…Ｕレジスタの示す番地からＵ＋Ｂの示す番地までのすべてのビットを反転させます（１を０に０を１にします）．

このルーチンはＬＩＮＥ文のＹ座標＝一定の（Ｘ軸に平行な）ＬＩＮＥを引くときに利用されます．

ⅲ）ＬＮＸＯＲ，ＬＮＸＡＮＤ，ＬＮＸＮＯＰ，ＬＮＸＥＯＲ

ＬＮＸＯＲ	＄Ｆ６３１～＄Ｆ６４０
ＬＮＸＡＮＤ	＄Ｆ６４７～＄Ｆ６５７
ＬＮＸＮＯＰ	＄Ｆ６５７
ＬＮＸＥＯＲ	＄Ｆ６５Ｅ～＄Ｆ６６Ｄ

機　能　Ｘレジスタの示すＶＲＡＭとＹレジスタの示すＲＡＭからＢ＋１バイト分ＯＲ，ＡＮＤをとり，ＶＲＡＭ（Ｘレジスタ）に書き込む．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ

入力情報　Ｘレジスタ　書き込むＶＲＡＭの先頭アドレス

Ｙレジスタ　ＶＲＡＭとＯＲ，ＡＮＤなどをとるためのデータが入っている先頭アドレス

Ｂレジスタ　書き込むバイト数－１

ワークエリア　ＶＡＣＣ　＄Ｄ４０９

解　説　これらのルーチン群は以下の機能を示します．

ＬＮＸＯＲ　…ＲＡＭとＶＲＡＭのＯＲをとってこれをＶＲＡＭに書き込みます．これをＢ＋１バイト続けます．このＶＲＡＭの先頭アドレスは　Ｘレジスタで，ＲＡＭの先頭アドレスは，Ｙレジスタで示されます．図３・５・37①参照

ＬＮＸＡＮＤ…ＲＡＭを反転し１を０に，０を１にした後ＶＲＡＭとＡＮＤをとりＶＲＡＭに書き込みます．これをＢ＋１バイト分続けます．このＶＲＡＭの先頭アドレスはＸレジスタで，ＲＡＭの先頭アドレスはＹレジスタで示されます．図３・５・37②参照

ＬＮＸＮＯＰ…何もしません

ＬＮＸＥＯＲ…ＲＡＭとＶＲＡＭのＥＯＲをとってＶＲＡＭに書き込みます．これをＢ＋１バイト分続けます．ＶＲＡＭとＲＡＭの先頭アドレスは他の２つと同じです．図３・５・37③参照

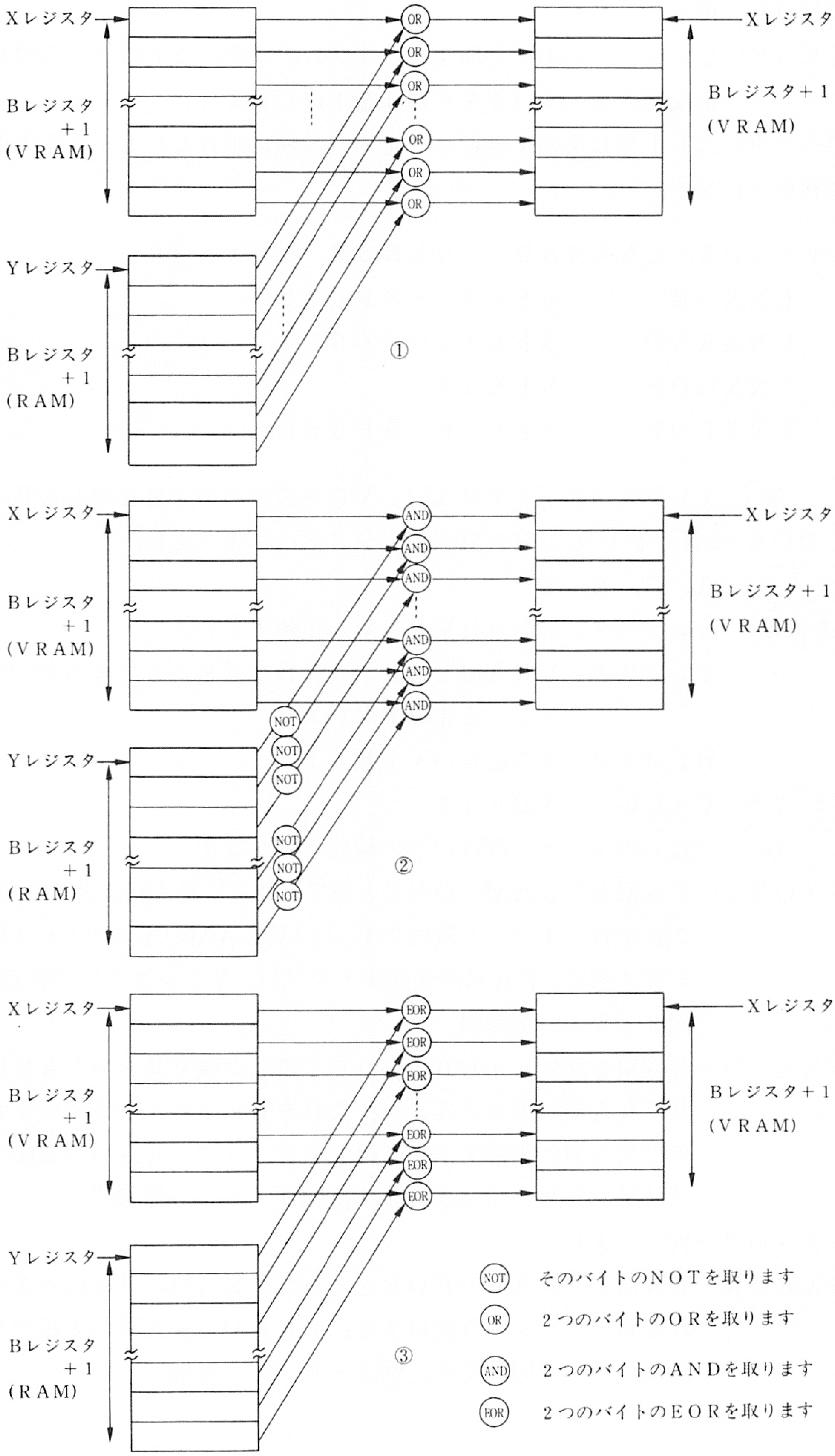

Xレジスタ
Bレジスタ
＋1
(VRAM)
Yレジスタ
Bレジスタ
＋1
(RAM)
Xレジスタ
Bレジスタ＋1
(VRAM)
OR
①
AND
NOT
②
EOR
③
NOT そのバイトのNOTを取ります
OR 2つのバイトのORを取ります
AND 2つのバイトのANDを取ります
EOR 2つのバイトのEORを取ります

図3・5・37

3）GADDR

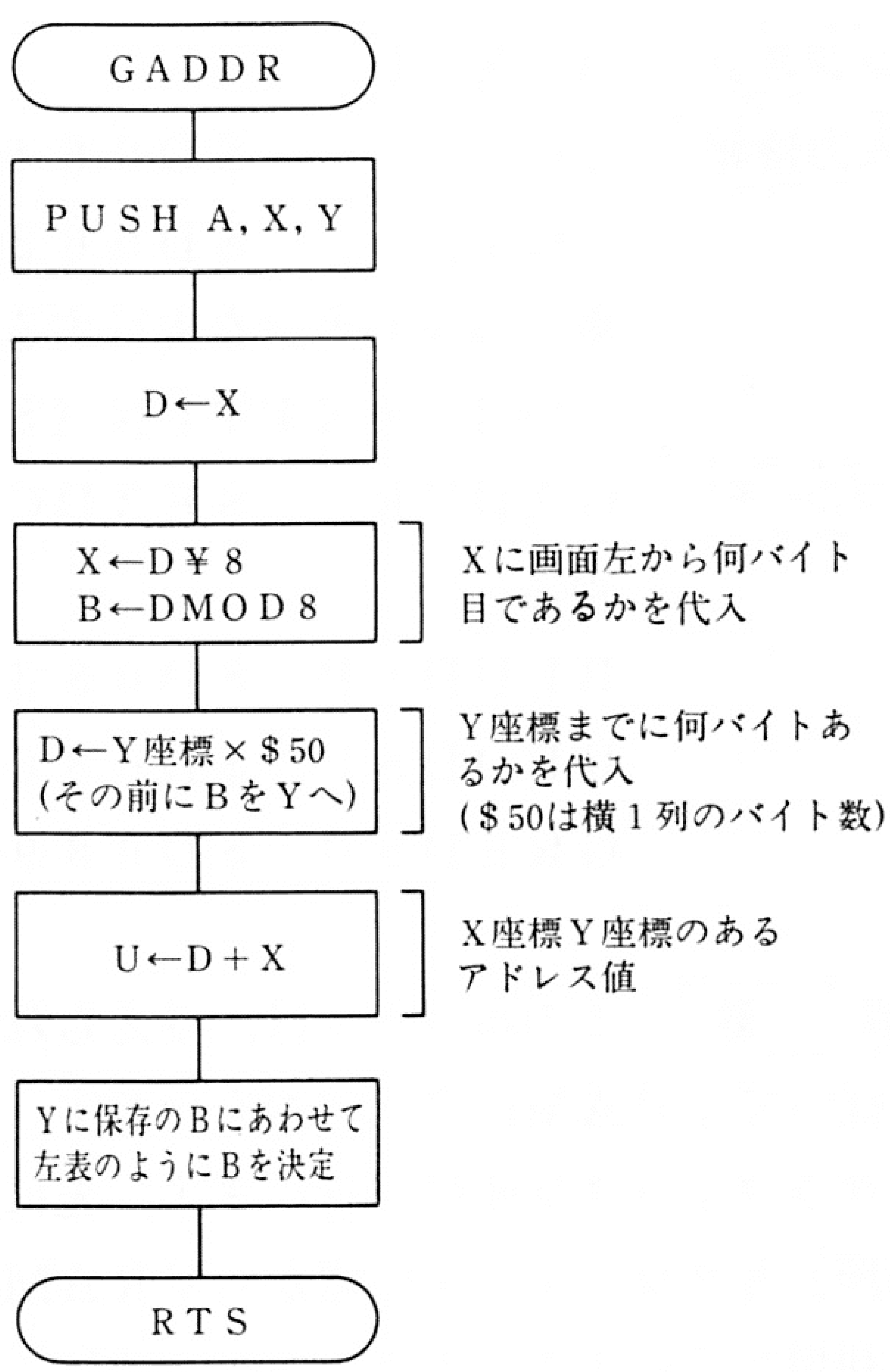
$E3DC～$E3F0

機　　能	グラフィック座標のX，Yに対してそれがVRAM上の何番地の何ビット目かを示す［(X座標，Y座標）→（VRAMアドレス，ビットマスク）変換］
レジスタ	A，B，X，Y，U（A，X，Yは保存）
入力情報	Xレジスタ　グラフィックX座標
	Yレジスタ　グラフィックY座標
出力情報	Uレジスタ　指定された点のVRAM上のアドレス(BLUEのみ)
	Bレジスタ　指定された点のビット のみを1にする（ビットマスク）
サブルーチン	GXOFFR　$E3F4　X=D ¥8，B=X MOD 8

図3・5・38

グラフィックX座標MOD8	Bレジスタ
00000000（0）	10000000
00000001（1）	01000000
00000010（2）	00100000
00000011（3）	00010000
00000100（4）	00001000
00000101（5）	00000100
00000110（6）	00000010
00000111（7）	00000001

図3・5・39



解　　説　前にも述べましたが，ＶＲＡＭ上の配列はグラフィック座標とはまるで違ったものです．ですからグラフィックを行うためには，必ずこの変換が必要なのです．注意してほしいのは，Ｂレジスタに返される値はビットマスクであること，すなわち，２進数で何ビット目かをさすのではなくて，図３・５・38のようにその指定されたビットに１を立てていることです．またこのルーチンでは青のＶＲＡＭのアドレスのみが返されますので他のアドレスを得たい時は＄４０００，＄８０００のオフセットを付けることが必要です．ＶＲＡＭとグラフィック座標とを理解できるようにフローチャートを示しておきます．

４）ＧＰＤＳＰ

ＧＰＤＳＰＣ　＄Ｅ４８Ｆ～＄Ｅ４ＡＢ
ＧＰＤＳＰＣ　＄Ｅ４９４（Ｘ，Ｙレジスタに座標入力）
ＧＰＤＳＰＲ　＄Ｅ４９７（Ｕ，Ｂレジスタにグラフィックアドレス）

機　　能　指定された点にファンクションコード，カラーに合わせて点を打ちます．

レジスタ　Ｂ，Ｘ，Ｕ

入力情報　．ＧＸ　＄Ｄ０６４～＄Ｄ０６５　グラフィックＸ座標
．ＧＹ　＄Ｄ０６２～＄Ｄ０６３　グラフィックＹ座標
※このルーチンを実行するまえにＧＰＤＳＰＩ（＄Ｅ４６９）あるいは，ＩＮＩＴＬＮ（＄Ｅ４５２）を呼んでおくこと

サブルーチン　ＧＡＤＤＲ　＄Ｅ３ＤＣ　ＶＲＡＭアドレスとドットパターンを得る
ＢＬＵＤＳＰ　＄Ｄ０８３
ＲＥＤＤＳＰ　＄Ｄ０８６
ＧＲＥＤＳＰ　＄Ｄ０８９
（以上３つ）ＩＮＩＴＬＮ参照　ドットを打つルーチン

解　　説　このルーチンは，与えられたファンクションコードとカラーコードに従ってＶＲＡＭに書き込むものです．書く位置の指定は，グラフィックＸ座標とグラフィックＹ座標で行います．先にあげたＶＲＡＭに書き込むルーチン〔２）参照〕とグラフィック座標からＶＲＡＭアドレスへ変換するルーチン〔３）参照〕を利用しています．このルーチンでは，ＧＡＤＤＲ〔３）参照〕で得られたＢレジスタをビットパターンとして(指定された点のみに１が立つ)，ＶＲＡＭに書き込みます．

5）GAXCHK

$E442〜$E447

機　能　指定されたグラフィック座標が画面上にないときにエラールーチンへ飛びます．

レジスタ　X，Y

入力情報　Xレジスタ　　グラフィックX座標
　　　　　Yレジスタ　　グラフィックY座標

サブルーチン　GAXCR　　$E388　グラフィック座標の範囲を調べCフラグを立てる．

解　説　このルーチンはグラフィック座標が範囲外であるかを調べます．具体的には，グラフィックX座標が0〜639，グラフィックY座標が0〜199の内部にあるか判断しています．

6）GXOFF，GXOFFR

GXOFF　　　$E3F2〜$E3FA
GXOFFR　　$E3F4

機　能　Dレジスタの8の剰余と8の整数除算を行います．

レジスタ　A，B，X

入力情報　GXOFFの場合
　　　　　Dレジスタ　　8の剰余と8で整数除算を行いたい数値
　　　　　GXOFFRの場合
　　　　　Dレジスタ　　8で整数除算を行いたい数値
　　　　　Xレジスタ　　8の剰余を行いたい数値

出力情報　GXOFFの場合
　　　　　Xレジスタ　　Dレジスタを8で整数除算した結果
　　　　　Bレジスタ　　Dレジスタの8の剰余
　　　　　GXOFFRの場合
　　　　　Xレジスタ　　Dレジスタを8で整数除算した結果
　　　　　Bレジスタ　　Xレジスタの8の剰余

サブルーチン　LSRD3　　$E3FB

解　　説　２進数での８の剰余は簡単です．例えばDレジスタに値が入っているときには，BレジスタのAND＃＄０７を実行すればよいのです．

7）ＬＳＲＤ３

＄Ｅ３ＦＢ～＄Ｅ４０１

機　　能　Dレジスタを８で整数除算を行います．

レジスタ　Ａ，Ｂ

入力情報　Dレジスタ　　８の整数除算を行いたい数値

出力情報　Dレジスタ　　Dレジスタを８で整数除算した結果

解　　説　このルーチンは，きわめて簡単でDレジスタを右へ３回シフトすることだけをしています．

8）ＧＰＤＳＰＩ

ＧＰＤＳＰＩ	＄Ｅ４６９	～　＄Ｅ４８Ｅ
ＧＰＤＳＩＲ	＄Ｅ４６Ｂ	
ＦＥＮＴＲＹ	＄Ｅ４６Ｅ	
ＬＥＮＴＲＹ	＄Ｅ４７１	

機　　能　グラフィックドット表示ルーチンのフックをイニシァライズします

レジスタ　Ａ，Ｂ，Ｘ，Ｕ

入力情報　※ＧＰＤＳＰＩエントリの場合

　.ＧＣＬ　　＄Ｄ０６０　グラフィックカラーコード

　.ＧＦ　　　＄Ｄ０６１　グラフィックファンクションコード

　※ＧＰＤＳＩＲ以下のエントリでは，

　Ａレジスタ　　グラフィックカラーコード

　Ｂレジスタ　　グラフィックファンクションコード

ワークエリア　..ＢＣＬ　　＄Ｄ０２Ｆ　バックカラーコード

　ＢＬＵＤＳＰ　＄Ｄ０８３～＄Ｄ０８５

　ＲＥＤＤＳＰ　＄Ｄ０８６～＄Ｄ０８８

　ＧＲＥＤＳＰ　＄Ｄ０８６～＄Ｄ０８８

機　能　このルーチンは，.ＧＣＬと.ＧＦで与えられたカラーとファンクションにしたがって，ＧＰＤＳＰ〔４）参照〕がＶＲＡＭに書き込むルーチン〔２）参照〕を呼びだすときのフック（ＢＬＵＤＳＰ～ＧＲＥＤＳＰ）を設定するルーチンです．フックに書き込むベクトルのデータとしてＧＰＤＦＮＣ（＄Ｅ４ＡＤ～）を参照しています．

ＧＰＤＳＩＲでエントリするときは，Ｄレジスタにカラーとファクションを設定しておきます．

ＦＥＮＴＲＹはPut Block 1で用いているエントリで，Ｘレジスタにジャンプテーブルの先頭アドレスを入力して呼び出します．

ＬＥＮＴＲＹでエントリするときは，Ｘレジスタにジャンプテーブルの先頭アドレス，Ｕレジスタにフックの先頭アドレス，Ｄレジスタにカラーとファンクションを入力情報として与えれば，別の場所にあるフックを設定することができます〔このルーチンは，１）で述べたＩＮＩＴＬＮの途中から始まる下位ルーチンで，ＩＮＩＴＬＮではこのエントリを用いてラインのフックを設定しています〕．

各画面ドット表示用フックには，ＧＰＯＲＡ・ＧＰＡＮＤなどへのジャンプ命令が書き込まれますが，カラーおよびファンクションコードに対するフックの設定の仕方は，ＣＤＳＰＩルーチンでの方法と同様です（３－４－７節参照）．

グラフィックドット表示用ジャンプテーブルＧＰＤＦＮＣの様子

アドレス	ジャンプベクトルの値	意　　味
＄Ｅ４ＡＤ～＄Ｅ４Ｂ０	ＧＰＯＲＡ(＄Ｅ４Ｃ１) ＧＰＡＮＤ(＄Ｅ４ＣＤ)	ＰＳＥＴファンクション(コード＝0)に対するベクトル
＄Ｅ４Ｂ１～＄Ｅ４Ｂ４	ＧＰＯＲＡ ＧＰＡＮＤ	ＰＲＥＳＥＴ(コード＝1)
＄Ｅ４Ｂ５～＄Ｅ４Ｂ８	ＧＰＯＲＡ ＧＰＮＯＰ(＄Ｅ４ＣＣ)	ＯＲ　(コード＝2)
＄Ｅ４Ｂ９～＄Ｅ４ＢＣ	ＧＰＮＯＰ ＧＰＡＮＤ	ＡＮＤ　(コード＝3)
＄Ｅ４ＢＤ～＄Ｅ４Ｃ０	ＧＰＥＯＲ(＄Ｅ４ＤＡ) ＧＰＮＯＰ	ＥＯＲ　(コード＝4)

※各欄の上段は，カラーコードの各ビットによって選択された原色画面に対応するフックの書込み情報を，下段は，カラーコードの各ビットによって選択されていない原色画面に対応するフックへの書込み情報をそれぞれ意味します．

3－6　ブロックデータ処理

本節では，Graphic Cursor, Get／Put Blockなどのように，多数のデータの集合体，すなわち，ブロックデータを扱う処理系を中心に話を進めていきます．

これらの処理系では，ビット操作を駆使したデータの変形，データのブロック転送，2次元的データを扱うための多重ループなどのテクニックが盛んに用いられており，なかなか興味深いところです．ＳＹＭＢＯＬ処理系などでは，座標変換を実行するために，スタックをワークエリアとして利用し，間接アドレッシングで参照するといった手法も使われています．ユーザの皆さんにも，これらの処理系から，次元を持った大量データの処理手順を学びとっていただきたいと思います．

なおＧＸＯＦＦ（3－5－5節参照）は，本来はグラフィックアドレスを求めるために利用されるサブルーチンですが，ブロックデータを扱う処理系においては，除数8の整数除算の商と余りを求める目的で再三にわたって利用されていますので，一言つけ加えておきます．

3－6－1　Graphic Cursor　　$１F　($F67D)

ユーザが，CRT画面を見ながら，キーボードによって指示した，10箇所以内の座標を読み取って，まとめてメイン側へ転送するコマンドです．サブシステムがこのコマンドを実行中の際，画面には十字形のグラフィックカーソルが現れるようになっており，コマンド名も，このことに由来します．

共有RAM上のパラメータ

アドレス	記　号	内　容	値
D383	CL	カーソルの表示色	0(黒)～7(白)
D384	N	要求する座標数	1～10
D385～D386	X	初期座標X（グラフィック座標，16ビット）	0～639 ($0000～$027F)
D387～D388	Y	初期座標Y(グラフィック座標，16ビット)	0～199 ($0000～$00C8)

共有RAM上の復帰情報

アドレス	記　号	内　容	値
D383～D384	X1	第1番目の指定座標X1(16ビット)	0～639 (グラフィックX座標)
D385～D386	Y1	第1番目の指定座標Y1(16ビット)	0～199 (グラフィックY座標)
⋮	⋮	⋮	⋮
D383+4(n−1)～D384+4(n−1)	Xn	第n番目の指定座標Xn(16ビット)	0～639 (グラフィックX座標)
D385+4(n−1)～D386+4(n−1)	Yn	第n番目の指定座標Yn(16ビット)	0～199 (グラフィックY座標)

主なサブルーチン

処理系外部

$E442　　GAXCHK　座標（X，Y）のチェック（エラー付き）

$FD6E　　KEYGET　キー入力（入力待ち付き）

＄Ｅ３ＤＣ	ＧＡＤＤＲ	座標（Ｘ，Ｙ）からグラフィックアドレス生成
＄Ｅ３ＦＢ	ＬＳＲＤ３	Ｄレジスタを右に３ビットシフト
処理系内部		
＄Ｆ８８０	ＧＣＢＤＲＹ	カーソルと画面の四隅(よすみ)との位置関係をチェック
＄Ｆ８ＥＣ	ＧＣＳＡＶＥ	カーソル位置の画面情報をスタックエリアに積む
＄Ｆ９４Ｄ	ＧＣＤＩＳＰ	グラフィックカーソル（十字形）を表示
＄ＦＡ２８	ＧＣＲＳＡＶ	積み上げられた画面情報を元通りに戻す
エラーエントリ		
＄Ｅ０ＤＥ	ＥＲＲ６５	座標数エラー（エラーコード＄４１）

補助ワークエリア

＄Ｄ０Ａ０ ～＄Ｄ０Ａ１	ＧＣＩＸ	カーソル初期Ｘ座標（共有ＲＡＭからの入力パラメータ）保存用....ＨＯＭＥ処理で使用
＄Ｄ０Ａ２ ～＄Ｄ０Ａ３	ＧＣＩＹ	カーソル初期Ｙ座標（同上）
＄Ｄ０Ａ４ ～＄Ｄ０Ａ５	ＧＣＸ	カーソル現在位置のＸ座標
＄Ｄ０Ａ６ ～＄Ｄ０Ａ７	ＧＣＹ	カーソル現在位置のＹ座標
＄Ｄ０Ａ８	ＧＣＣＯＤＥ	キー入力コード保存用
＄Ｄ０Ａ９	ＧＣＣＬ	カーソル表示色のカラーコード
＄Ｄ０ＡＡ	ＧＣＳＣＯＮ	シフトカウントセットエリア（サブルーチン用のパラメータ）
＄Ｄ０ＡＢ ～＄Ｄ０ＡＣ	ＧＣＳＤＯＴ	シフトドット数.. 数字キー[1]～[9]および[0]により設定される，カーソル移動処理時のドット数
＄Ｄ０ＡＤ	ＧＣＢＣＯＮ	バイトカウント....設定されるが使用されないエリア
＄Ｄ０ＡＥ	ＧＣＮＣＯＮ	メイン側から要請された座標数（ダウンカウントに利用される）
＄Ｄ０ＡＦ	ＧＣＭＣＯＮ	要請された座標数（設定されるが未使用）
＄Ｄ０Ｂ０ ～＄Ｄ０Ｂ２	ＧＣＣＵＲＳ	カーソルパターンレジスタ（カーソル中心座標の位置により，＄ＦＦＦＦ８０～＄０１ＦＦＦＦのパターンデータを保持する）
＄Ｄ０Ｂ３	ＧＣＳＰＴＲ	スタックエリア用ポインタ

〜＄D0B4		
＄D0B5	GCRPTR	共有RAM結果格納用ポインタ
〜＄D0B6		
＄D0B7	GCBLUE	カーソルのカラーコード“1”に対応する色には
＄D0B8	GCRED	＄FF，“0”に対応する色には＄00「格納
＄D0B9	GCGREE	され，カーソル表示の際に参照される
＄D0B一	GCBITM	ビットマスク退避場所
＄D0BB	GCWOR	ビットマスク反転データ（GCBITMの反転）
＄D0BC 〜＄D0BD	GCWORK	VRAM実効アドレス（Uレジスタ）の退避場所
＄D0BE	GCFLG3	リターンフラグ（リターンキー⏎入力の直後にセット，その状態からカーソルが動けばクリアされる．カーソル分岐の判断に利用される）
＄D0BF	GCFLG5	未使用
＄D0C0 〜＄D0C2	GCCRSI	カーソルパターン初期値保存用レジスタ （HOME処理で使用）
＄D0C3 〜＄D0E9	GCAREA	カーソル現在位置に対応する画面データの退避場所（ヘッダ6バイト，各色11バイトずつの合計39バイト）
＄D0EA 〜＄D20B	GCSARE	画面情報退避用のスタックエリア（実際に使用されるのは，要請された座標数＊39バイト）

解　説　Graphic Cursorは，ユーザがグラフィックカーソル（十印）を操作することによって指定した座標を読み取って，メイン側に転送するという機能を持ったコマンドです．このコマンドが実行されると画面に＋印のカーソルが現れますが，この時，もともとカーソルの位置に存在していた画面のパターンデータを退避させておき，コマンド実行終了時には，再び画面上の元通りの位置に復帰させるという処理ステップが必要になります．Graphic Cursor処理系はメインループの他に上記の退避・復帰を行う，GCSAVE・GCRSAV，およびカーソル表示を行うGCDISP，カーソルと画面の四隅との位置関係（境界条件）を判定するGCBDRYなどのサブルーチンによって構成されています．

一方，ユーザのキー入力による操作を実行するために，メインループ内には各キー入力に対応する処理ステップが主としてサブルーチンの形で存在します．表3・6・1に，これらの処理ステップのエントリをあげますが，これらの処理

ステップは，Graphic Cursor 設定環境下でのみ呼び出すことが可能であり，しかも，先ほどの４つのサブルーチン（ＧＣＳＡＶＥ以下）の助けを借りてはじめて有効な仕事ができるという点に注意して下さい．

表３・６・１　キー入力に対応する処理

入力キーコード （16進の値）	エントリ （ラベル名）	機　　能
→ 右向き矢印 （＄１Ｃ）	＄Ｆ８０５ （ＧＣＳＦＴＲ）	カーソルを右に１ステップ （ＧＣＣＯＮＴ分）移動
← 左向き矢印 （＄１Ｄ）	＄Ｆ８４Ａ （ＧＣＳＦＴＬ）	カーソルを左に１ステップ 移動［以上，ＧＣＸ更新］
↑ 上 （＄１Ｅ）	＄Ｆ７Ｅ０ （ＧＣＵＰ）	カーソルを上に１ステップ （ＧＣＣＯＮＴ分）移動
↓ 下 （＄１Ｆ）	＄Ｆ７ＥＣ （ＨＣＤＷ）	カーソルを上に１ステップ 移動［以上，ＧＣＹ更新］
ＳＨＩＦＴ＋→ （＄０６）	＄Ｆ７ＦＣ （ＧＣＲ20）	カーソルを右に20ドット 移動［ＧＣＸ更新］
ＳＨＩＦＴ＋← （＄０２）	＄Ｆ８３Ｆ （ＧＣＬ20）	カーソルを左に20ドット 移動［ＨＣＸ更新］
ＳＨＩＦＴ＋↑ （＄１９）	＄Ｆ７Ａ８ （ＧＣＵＰ20）	カーソルを上に20ドット 移動［ＧＣＹ更新］
ＳＨＩＦＴ＋↓ （＄１Ａ）	＄Ｆ７Ｂ５ （ＣＣＯＤＷ20）	カーソルを下に20ドット 移動［ＧＣＹ更新］
ＨＯＭＥ （＄０Ｂ）	＄Ｆ７ＣＦ （ＧＣＨＯＭＥ）	カーソルを初期位置に戻す ［ＧＣＩＸ，ＧＣＩＹ利用］
数字キー０～９ （＄30～＄39）	＄Ｆ７０７ （特にラベルなし）	キーの数値をＨＢＢＯＮＴ にセットする（０→10）
ＲＥＴＵＲＮ （＄０Ｄ）	＄Ｆ７１８ （ＧＣ０００４）	現在のカーソル位置をユー ザ指定座標として読み取る

ここに示したエントリのうち，キー入力コード＄１９～＄１Ｆ（ＳＨＩＦＴ＋↑～↓）については，ジャンプベクトル方式が用いられており，＄Ｆ７９Ａ～＄Ｆ７Ａ７番地にジャンプテーブルが用意されております．

図3・6・1　GCURS処理系のアルゴリズム（5の1）ブロック構成

F67A　GCERR1　ERR65へ分岐（インバリッドコマンドエラー）

メインルーチン

F67D　GCURS　グラフィックカーソルエントリ
F6F6　GCLOOP　メインループ先頭

F718　GCRTKY　リターンキー処理
F740　GCIRQ　アボード処理（キャンセル）
F746　GCRTS　GCURSコマンドのエンディングルーチンエントリ

F756　GCMOVE　カーソル移動キーの処理

F766　キーコードが$02なら　GCL20　をコール
F76F　キーコードが$06なら　GCR20　をコール
F778　キーコードが$0Bなら　GCHOMEをコール
F781　その他のキーコードなら，ジャンプテーブルにより各処理系をコール
F78B　カーソル移動処理エンディング（境界条件判定，画面情報退避，カーソル表示）

F79A　GCTABL　GCURSカーソル移動処理用ジャンプテーブル
〜　※キーコード$19（GCUP20）から$1F（GCDW）に対する
F7A7　処理用のジャンプベクトル群

F7A8　GCUP20　カーソルを上に20ドット移動
F7B5　GCDW20　カーソルを下に20ドット移動
F7CF　GCHOME　カーソルを最初の位置まで戻す
F7E0　GCUP　カーソルを上に1ステップ移動
F7EC　GCDW　カーソルを下に1ステップ移動
F7FC　GCR20　カーソルを右に20ドット移動
F805　GCSFTR　カーソルを右に1ステップ移動
F83F　GCL20　カーソルを左に20ドット移動
F84A　GCSFTL　カーソルを左に1ステップ移動

GCX，GCYおよびGCCURS（カーソルパターンレジスタ）の値を更新するのみ

F880　GCBDRY　境界条件の判定ルーチン（画面の四隅にカーソルがかかるかどうか）

F8EC　GCSAVE　画面情報の退避ルーチン（Xレジスタにスタックエリア先頭を入力）
F910　GCSAV1　退避実行ルーチン

F94D　GCDISP　グラフィックカーソル表示ルーチン
F986　GCDOR　点灯（OR）処理による表示
F9D2　GCDAND　消灯（AND）処理による表示

FA28　GCRSAV　画面情報の復帰ルーチン
FA4C　GCRSV1　復帰実行ルーチン

図3・6・1　GCURS処理系のアルゴリズム（5の2）GCURS初期設定部のフローチャート

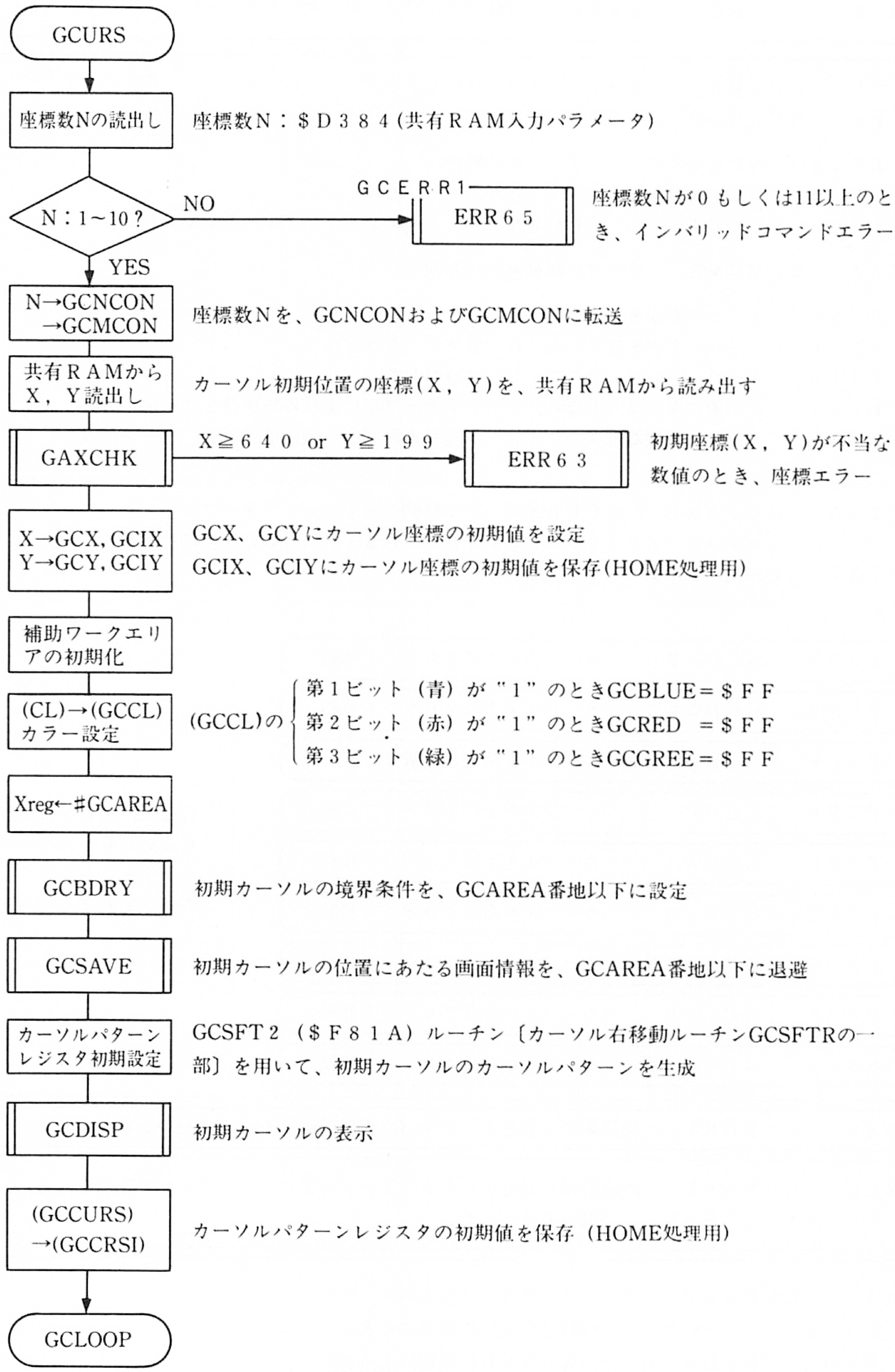


図3・6・1　GCURS処理系のアルゴリズム（5の3）メインループのフローチャート

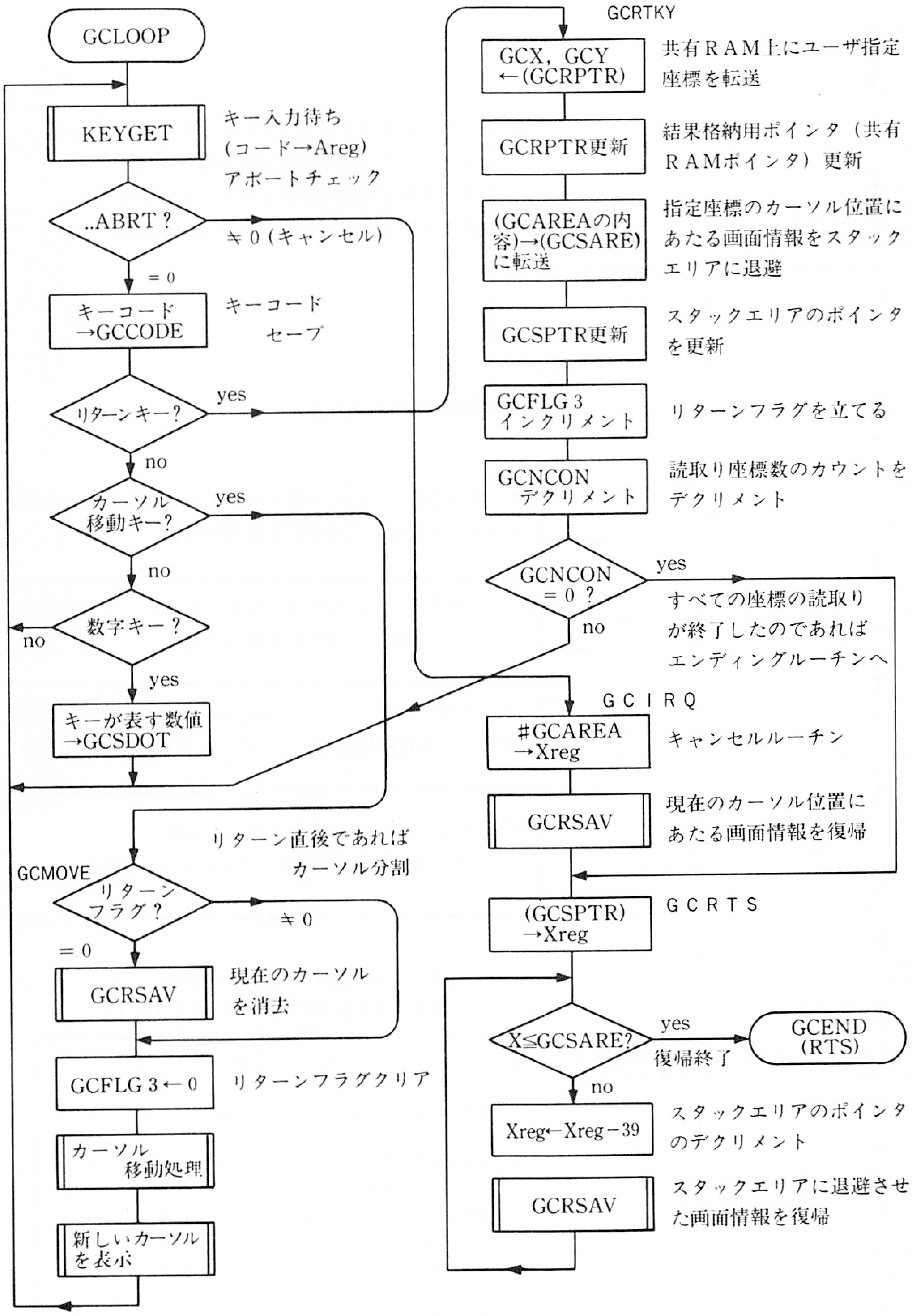
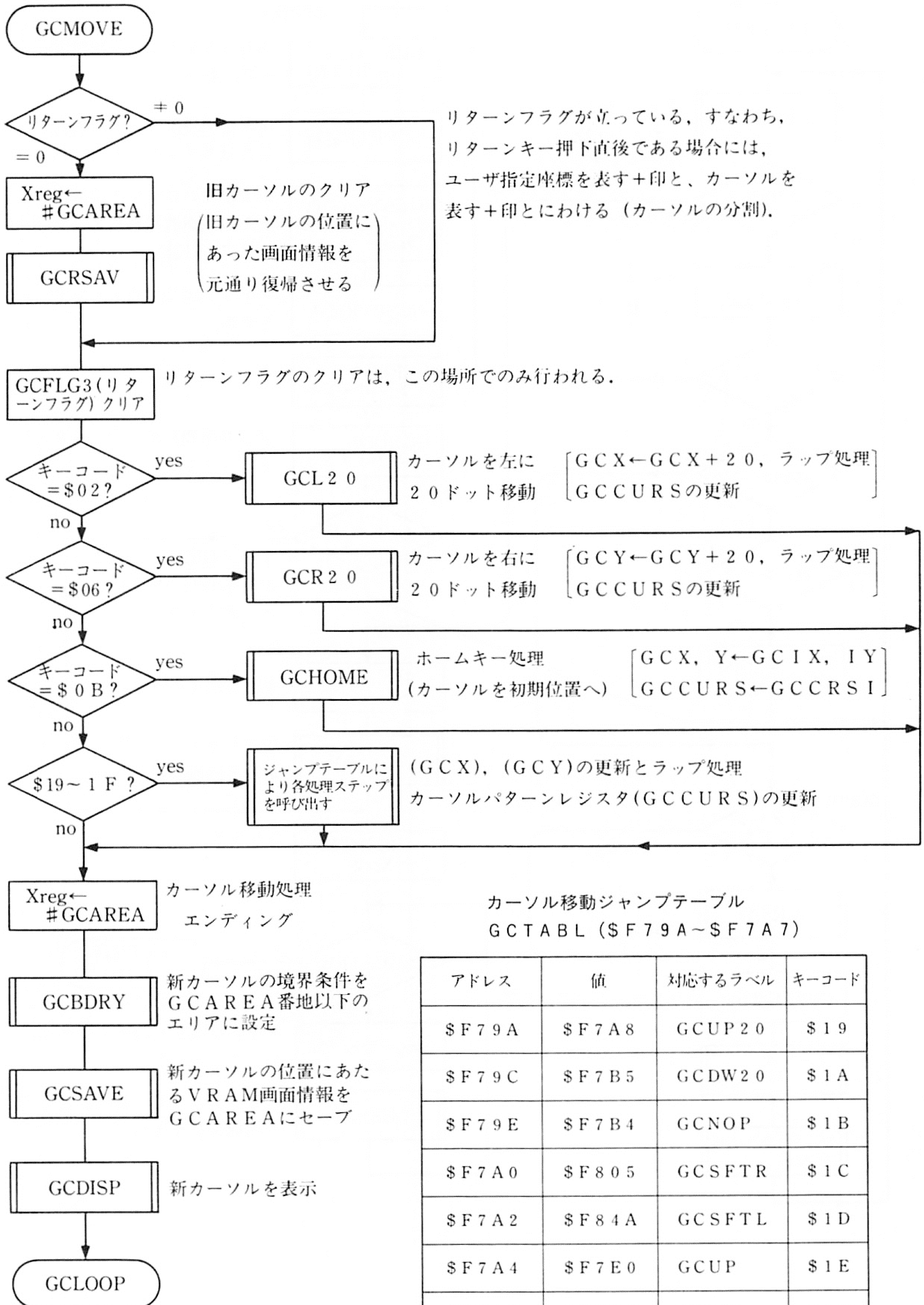


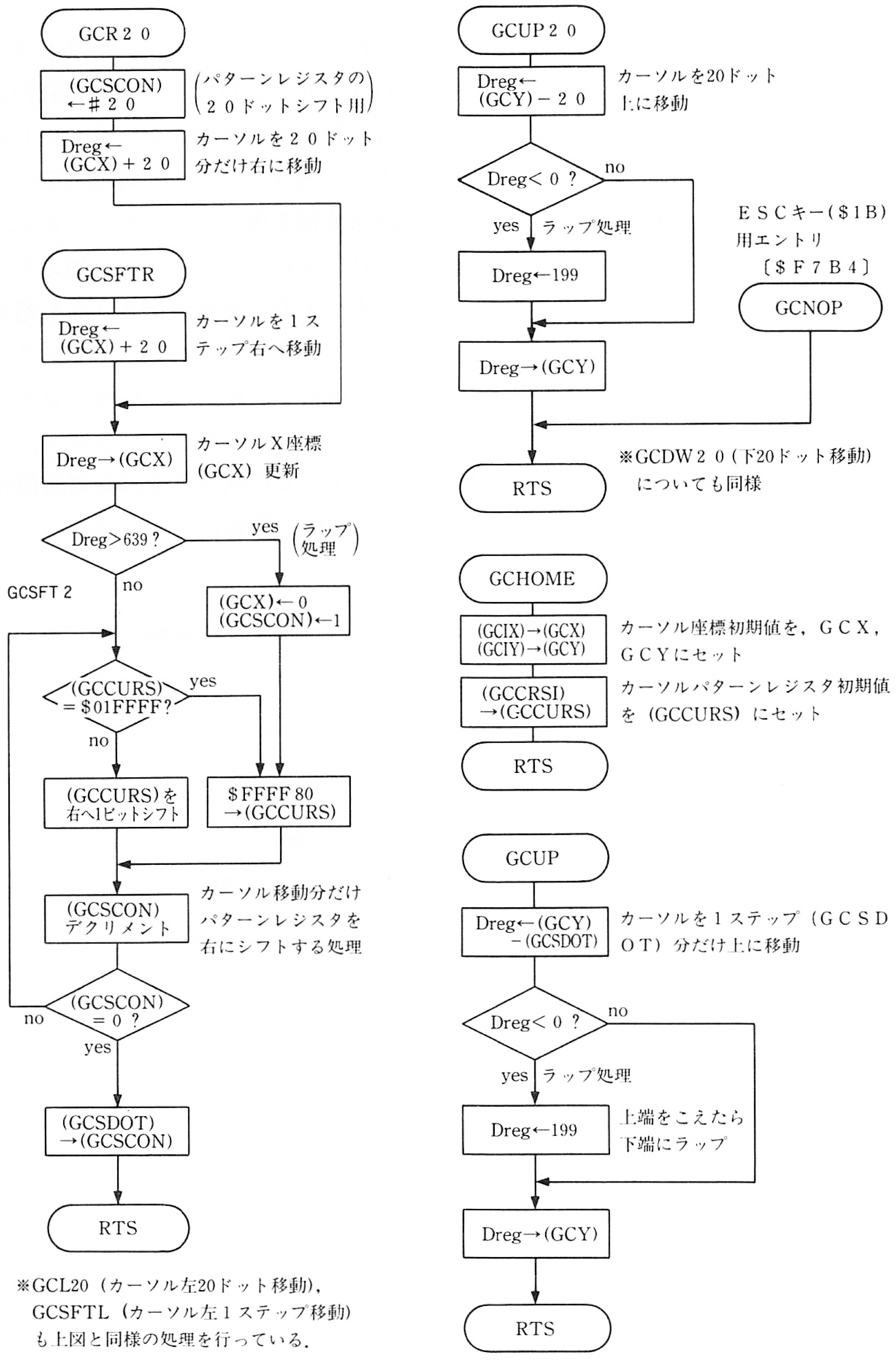
図3・6・1　GCURS処理系のアルゴリズム（5の4）カーソル移動キー処理（全体）



カーソル移動ジャンプテーブル
GCTABL（$F79A～$F7A7）

アドレス	値	対応するラベル	キーコード
$F79A	$F7A8	GCUP20	$19
$F79C	$F7B5	GCDW20	$1A
$F79E	$F7B4	GCNOP	$1B
$F7A0	$F805	GCSFTR	$1C
$F7A2	$F84A	GCSFTL	$1D
$F7A4	$F7E0	GCUP	$1E
$F7A6	$F7EC	GCDW	$1F

図3・6・1 GCURS処理系のアルゴリズム（5の5）カーソル移動キー処理（各ステップごと）

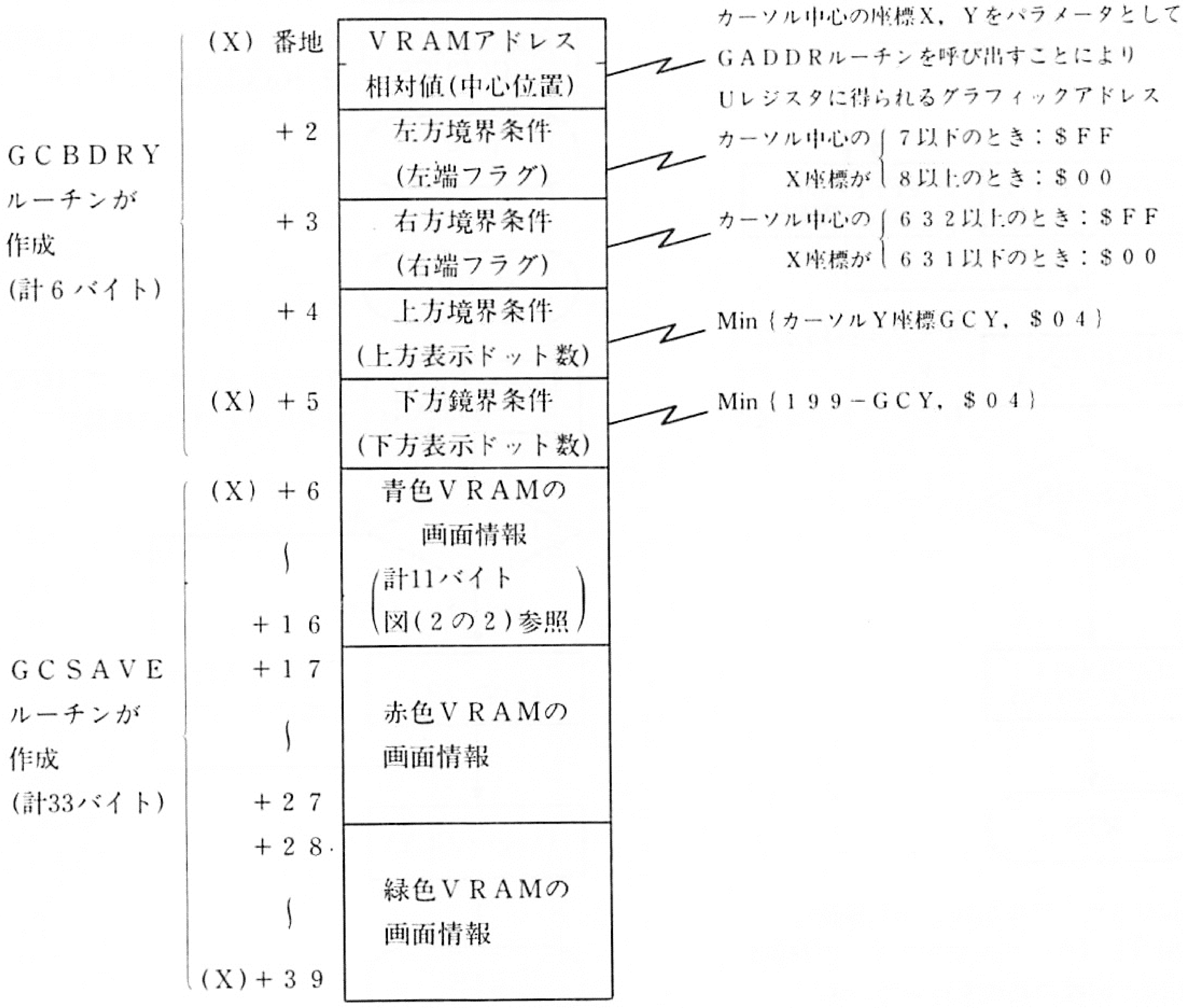


それでは，ＧＣＵＲＳ（Graphic Cursor）処理系メインループ全体のアルゴリズムを図３・６・１に示します．

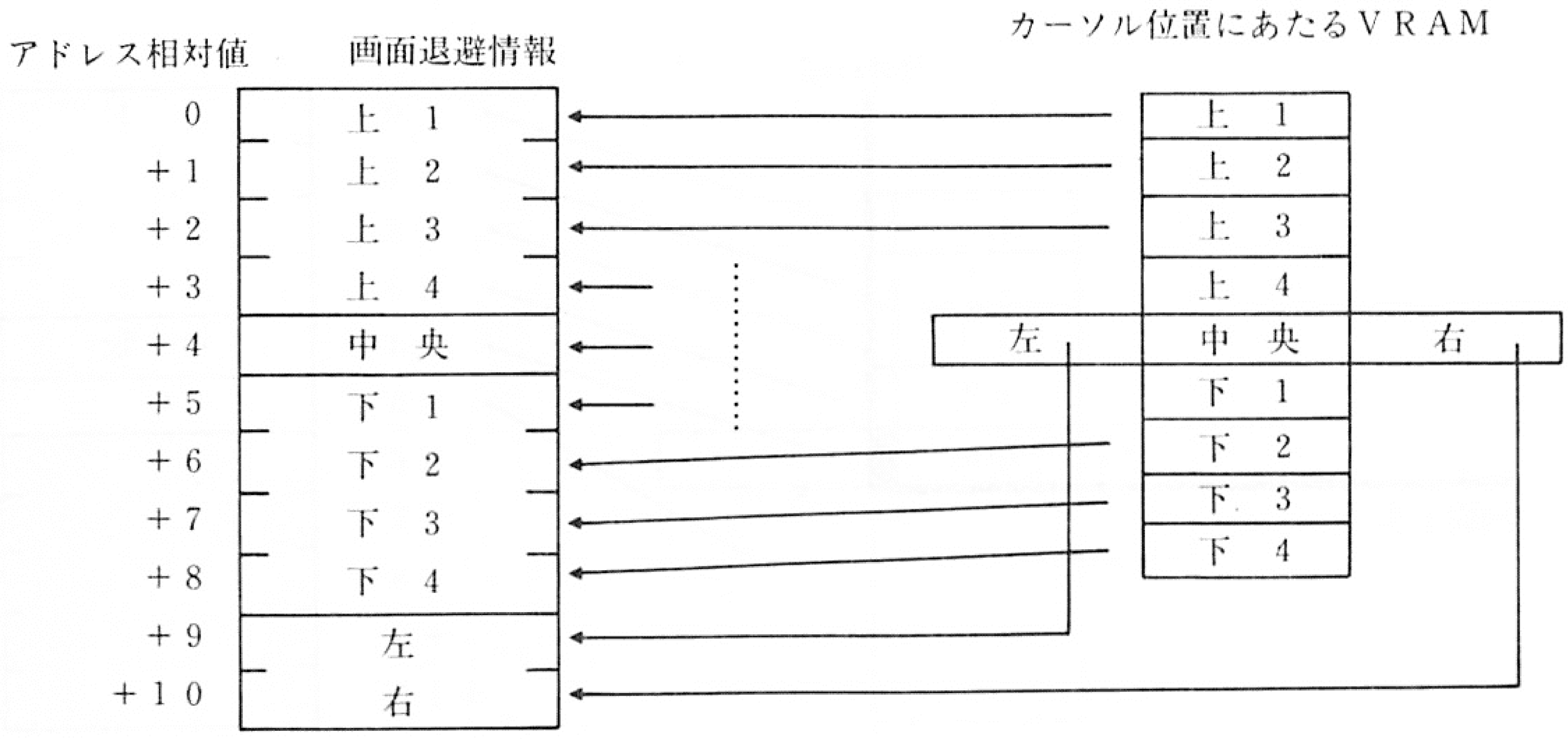
各カーソル移動ステップでは，カーソル座標ポインタＧＣＸ，ＧＣＹおよびカーソルパターンレジスタＧＣＣＵＲＳ（＄Ｄ０Ｂ０～＄Ｄ０Ｂ２）の更新のみを行うだけで，最終的にＧＣＢＤＲＹおよびＧＣＤＩＳＰの助けを借りてはじめてカーソルの移動・表示が完了します．たとえば，[→]（＄１Ｄ）の処理ステップでは，あらかじめ，今までのカーソル位置の画面情報を復帰させておいてから，ＧＣＸ（Ｙ座標）に，ＧＣＳＤＯＴ（＄Ｄ０ＡＢ～＄Ｄ０ＡＣ）の値を加えることによってカーソル移動を実現しております（座標Ｘの計算結果が６３９を超えると，ラップして，ＧＣＸ＝０を代入する）．同時に，カーソルパターンレジスタＧＣＣＵＲＳ（＄Ｄ０Ｂ０～＄Ｄ０Ｂ２）の値も更新し，最後にＧＣＢＤＲＹ，ＧＣＳＡＶＥ，ＧＣＤＩＳＰを呼び出してメインループへと戻っていきます．

ここで，ＧＣＢＤＲＹおよびＧＣＳＡＶＥによって作成される画面退避情報の形状，並びに退避される画面情報の種類と退避場所の関係について図示しておきます．

図３・６・２　ＧＣＢＤＲＹ，ＧＣＳＡＶＥがスタックエリアに作成する画面退避情報の形状（２の１）



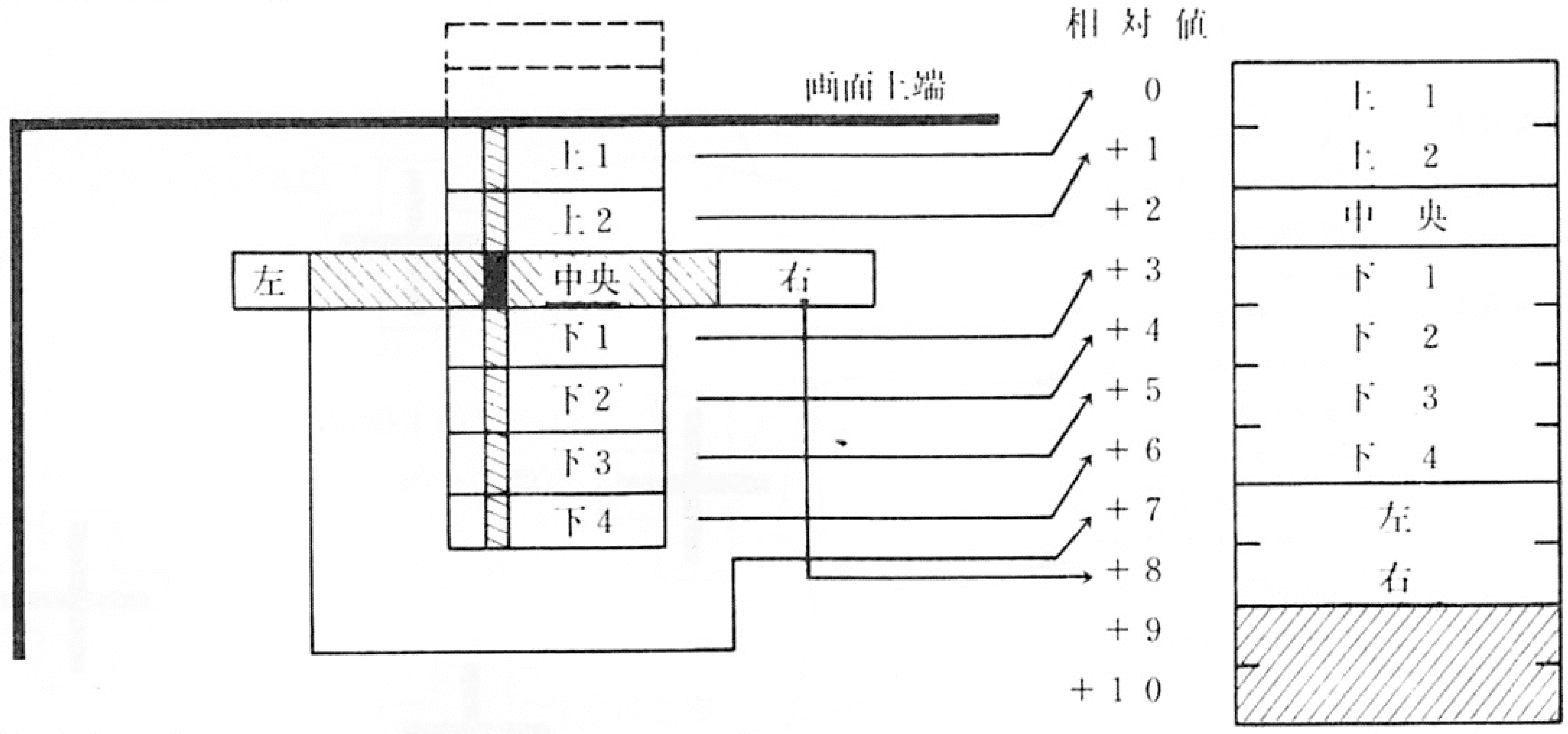
各色VRAM画面情報の詳細



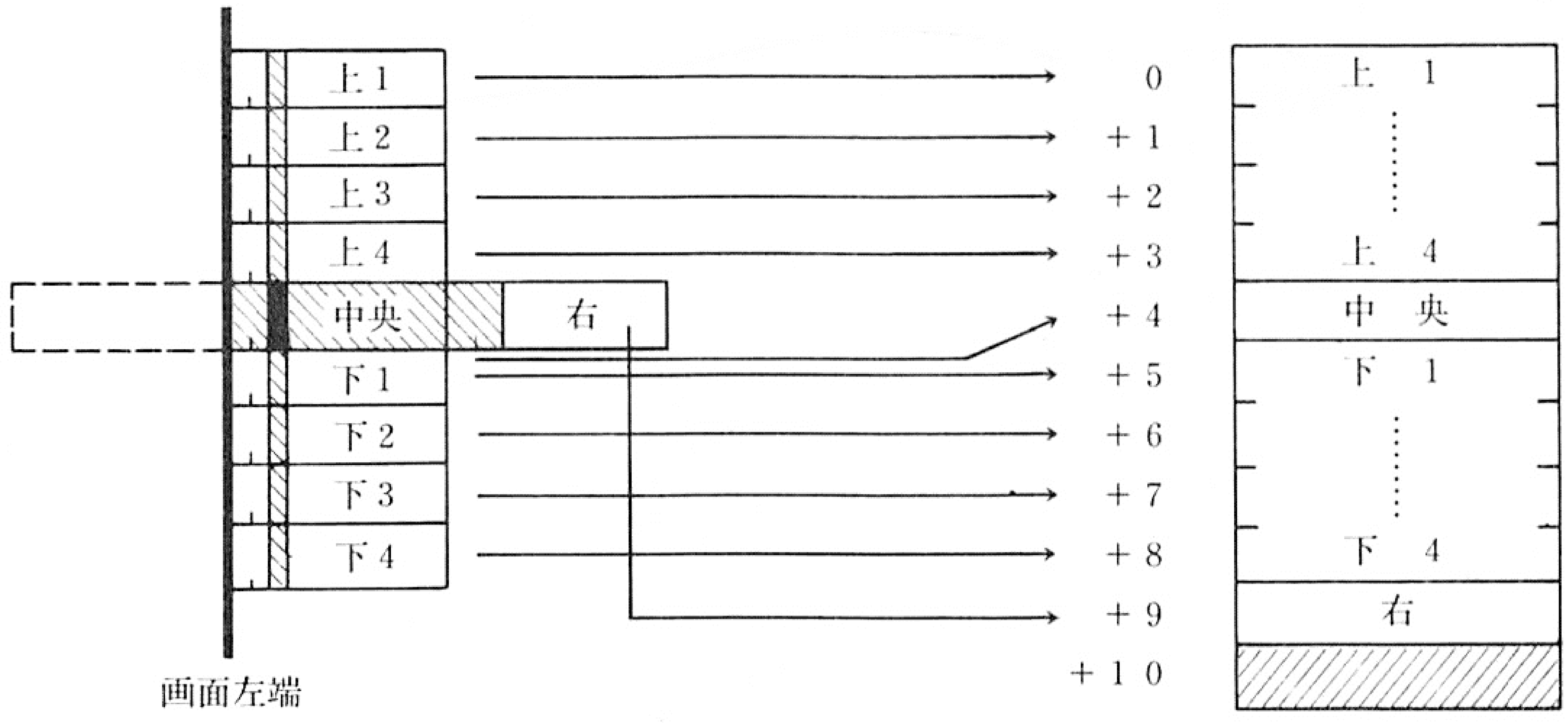
※境界条件によって，退避が不必要なものは順次省略されていき間が詰められて格納される〔次ページ図3・6・2（2の2）参照〕。

図3・6・2　GCBDRY, GCSAVEがスタックエリア上に作成する画面退避情報の形状（2の2）

①　例えば，Y座標が2のとき



②　例えば，X座標が3のとき



③ 例えば，$\left\{\begin{array}{l} X=639 \\ Y=199 \end{array}\right\}$ のとき

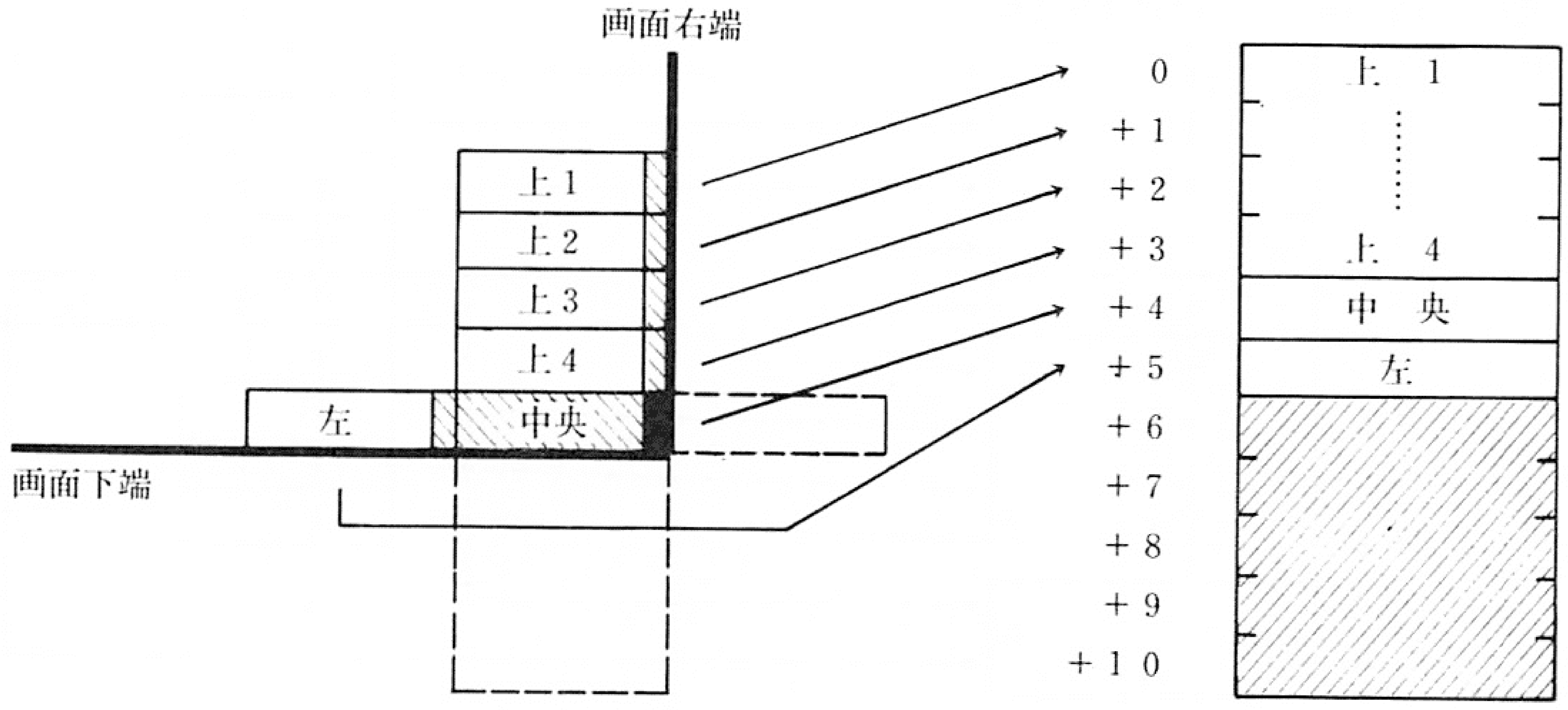
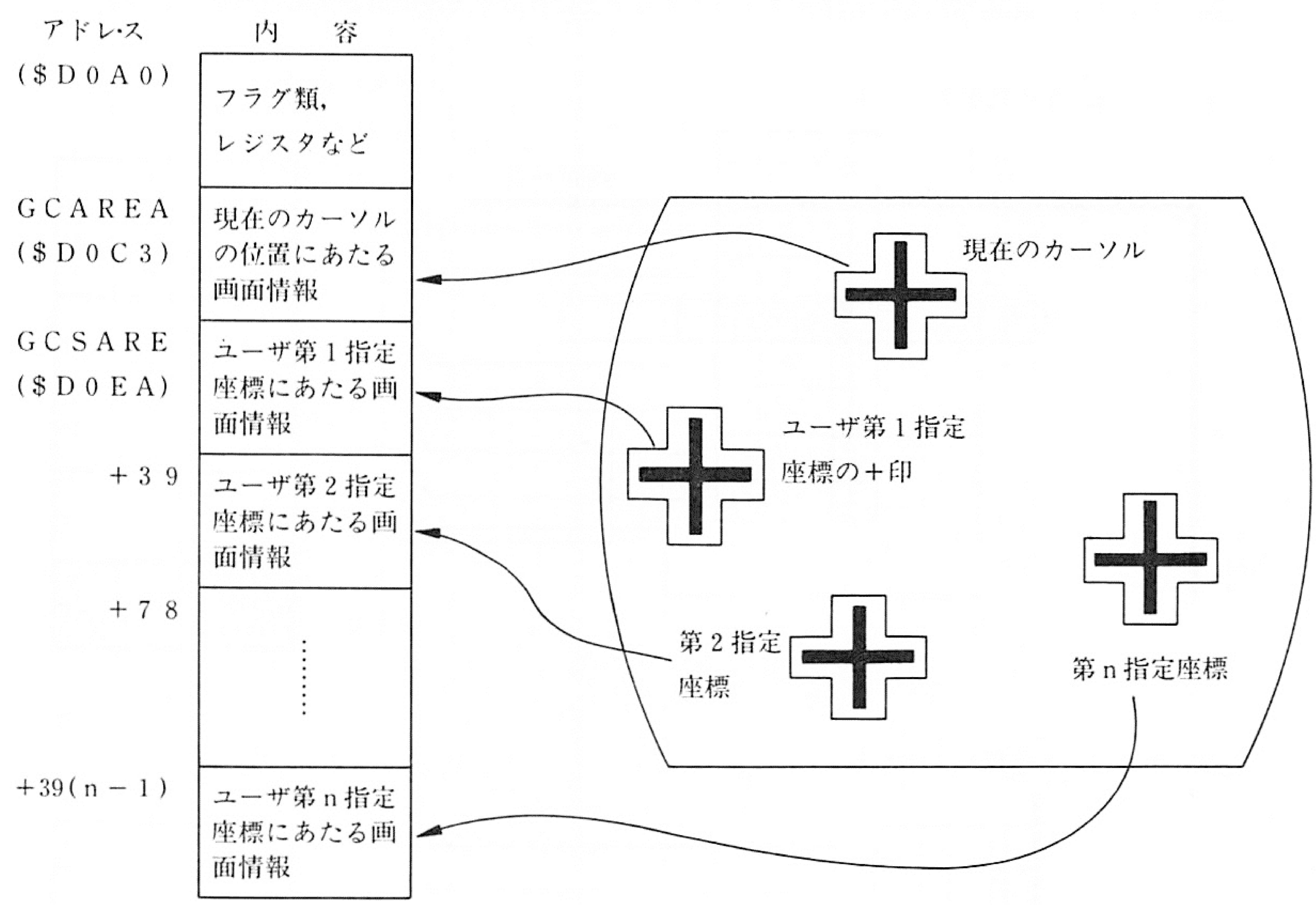
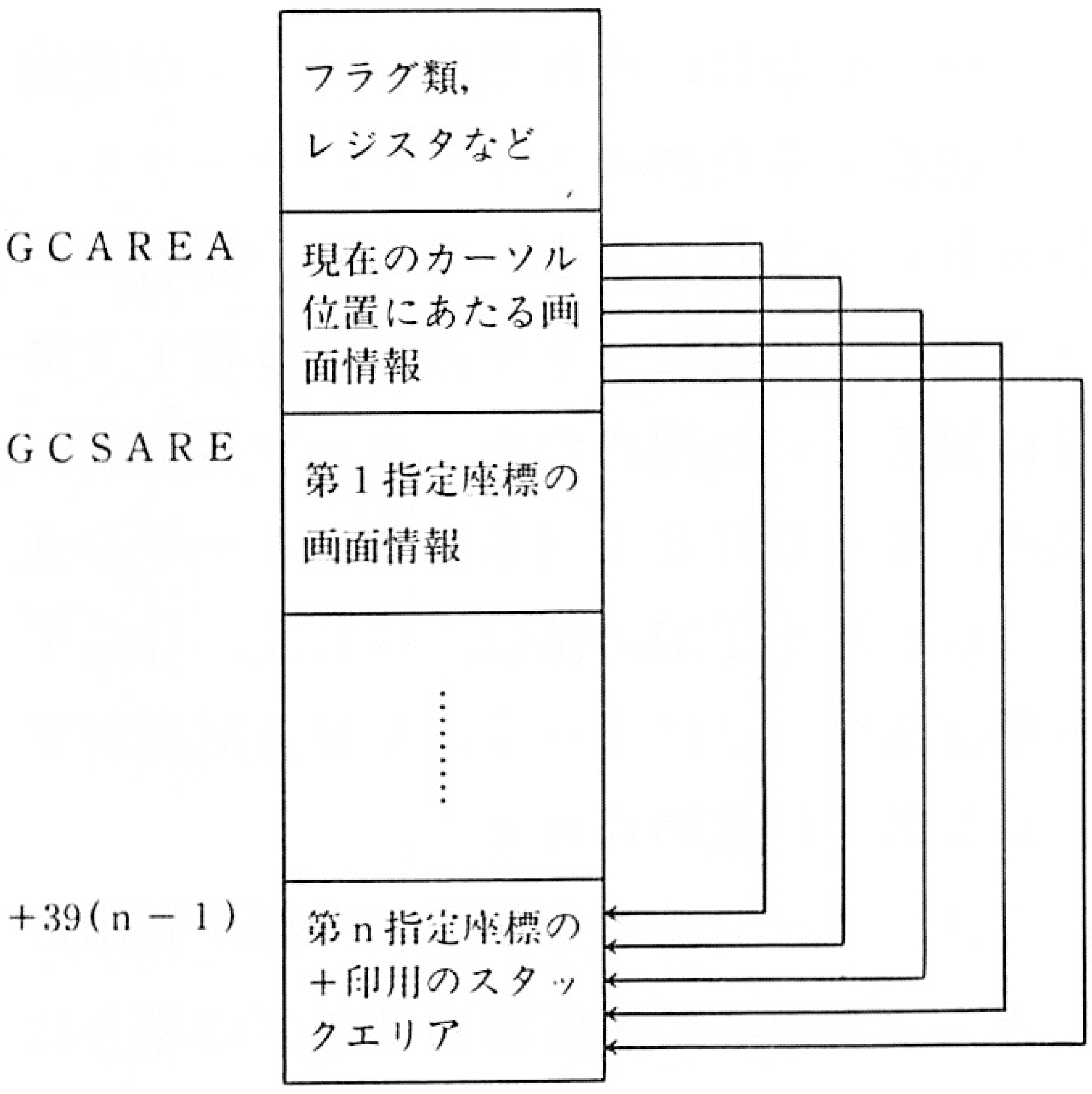


図3・6・3 画面退避情報の種類と退避場所の関係



画面退避情報の移動

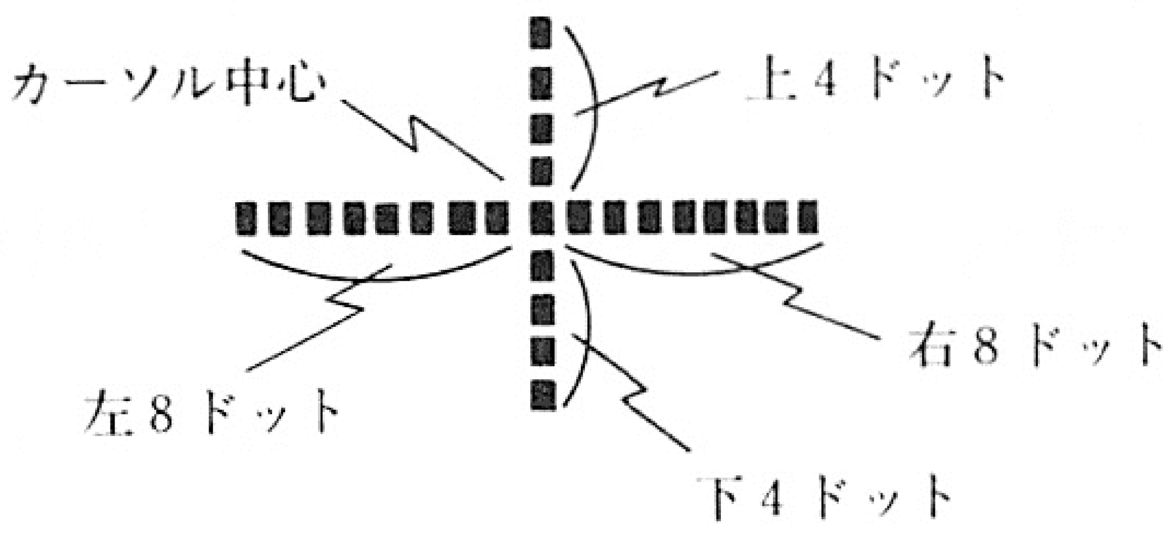


★リターンキーが押されると，GCRTKYルーチンによって、現在のカーソルの位置にあたる画面退避情報が、第n指定座標用のスタックエリアにブロック転送される。こののち、カーソル移動処理が実行されると、カーソル用の+印と、第n指定座標を表す+印の2つに分割される。

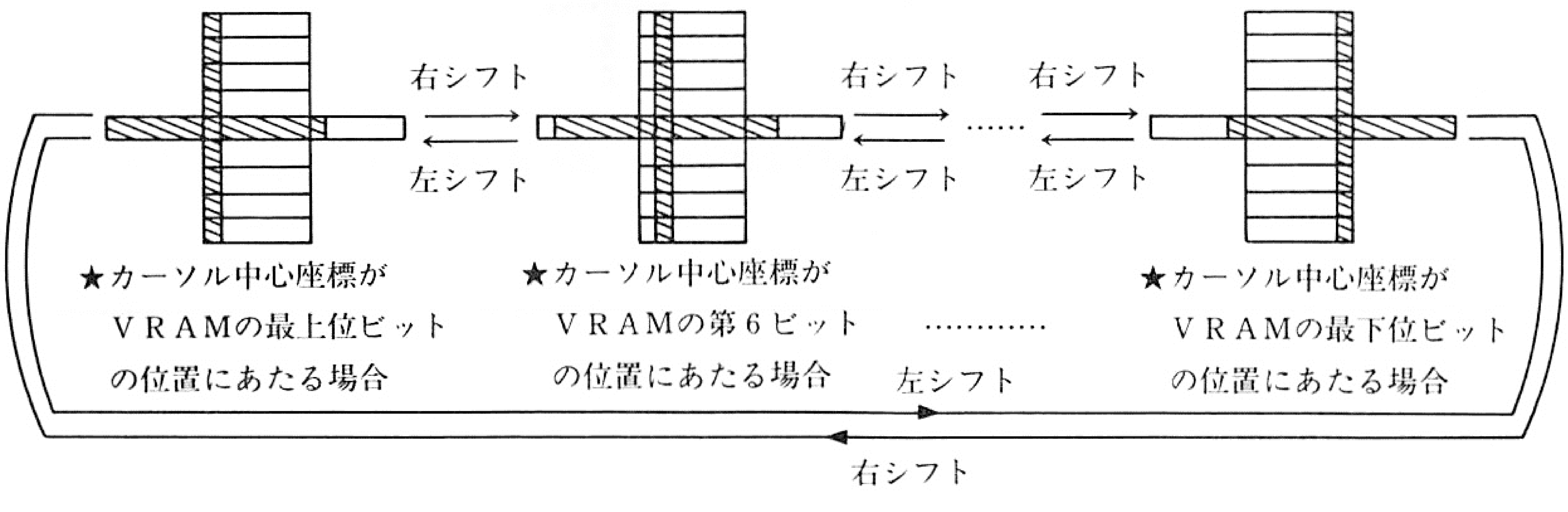
グラフィックカーソル（+印）とカーソルパターンレジスタ（GCCURS）

グラフィックカーソルというのは，右図のように，中心とその上下各4ドットずつ，および左右8ビットずつの計25ドットから構成される十字形でできています．このカーソルが表示されると,横方向3バイトと,上下4バイトずつの計11バイトのVRAMエリアを食いつぶすことになりますが，カーソルのパターンは下図のように，VRAMの各バイトの区切りとカーソル中心との位置関係によって8通り考えられます．

グラフィックカーソルのドット構成



グラフィックカーソルの移動に伴うカーソルパターンの変化（VRAMとの位置関係）

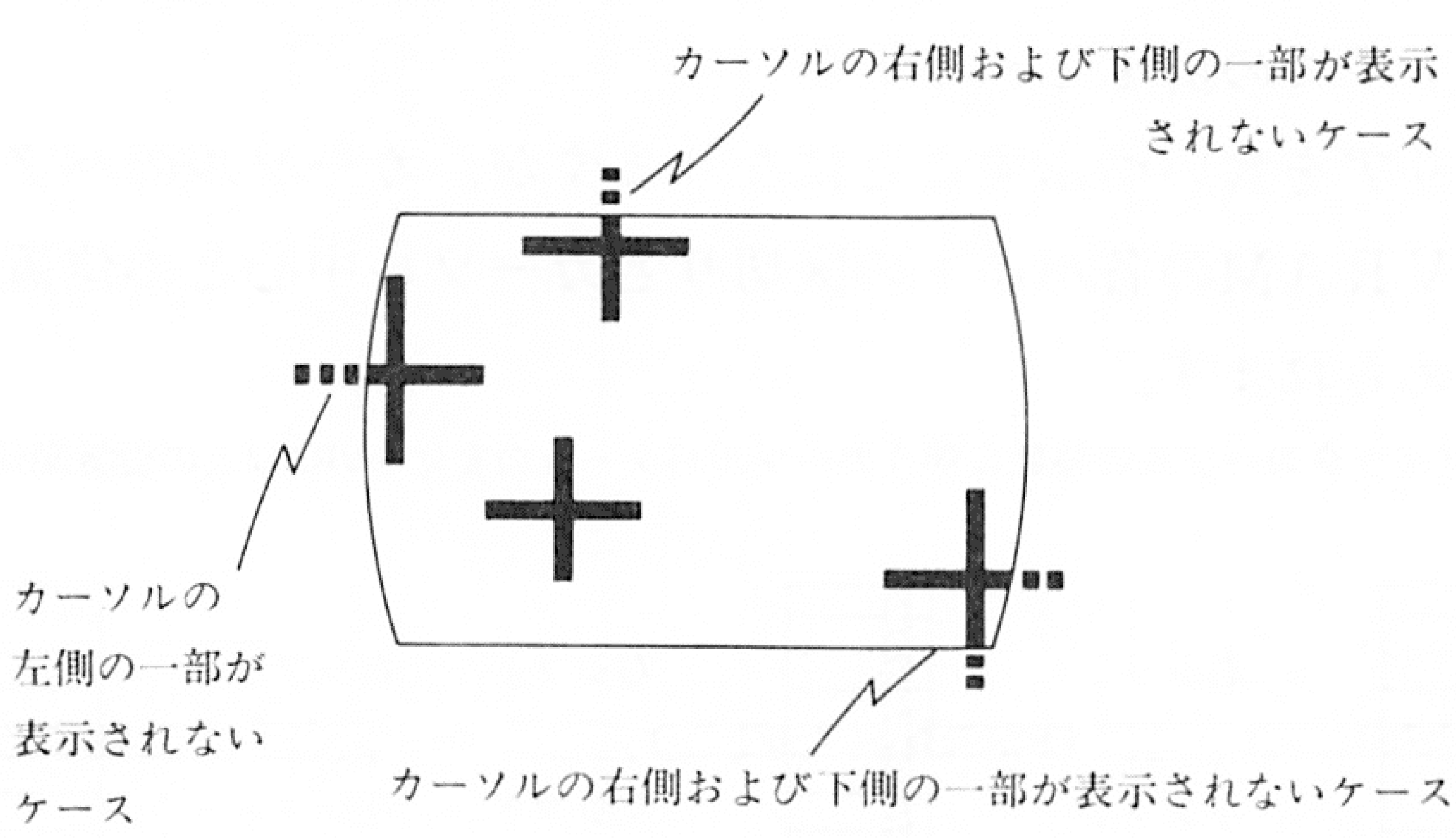


つまり，カーソル中心の座標によって，カーソルの横棒にあたる3バイトは$ＦＦＦＦ80〜$01ＦＦＦＦという具合に，カーソルの縦棒にあたる上下の各4バイトは$80〜$01という具合に，それぞれパターンが変化します．このうち，前者のパターンを記憶するためのエリアが，カーソルパターンレジスタＧＣＣＵＲＳ（$Ｄ0Ｂ0〜$Ｄ0Ｂ2）であり，後者のパターンを記憶するのが，グラフィックカーソル用ビットマスクＧＣＢＩＴＭ（$Ｄ0ＢＡ）なのです．また，ＨＯＭＥキー処理のため，カーソルパターンレジスタ初期値を保存しておく場所，ＧＣＣＲＳＩ（$Ｄ0Ｃ0〜$Ｄ0Ｃ2）も用意されております．なお，ＧＣＢＩＴＭの値は，ＧＣＸ，ＧＣＹを読み出してＧＡＤＤＲルーチンを呼び出すことによって，ＶＲＡＭ相対アドレス（Ｕレジスタ）とともに，Ｂレジスタに求められます．

一方，カーソルの中心座標によっては，カーソルの十字形の一部が表示されないケースがあります．例えば，カーソル中心のＸ座標が7以下の場合には，カーソルの左側の一部が，またカーソル中心のＹ座標が3以下の場合にはカーソルの上側の一部が，それぞれ表示されません．カーソル中心のＸ座標，Ｙ座標が，それぞれ632以上，196以上の場合も同様です．

このようなケースの判定を行うためのサブルーチンがＧＣＢＤＲＹ（境界条件判定ルーチン）で，Ｘレジスタの示す番地から，ＶＲＡＭ相対アドレス（2バイト），左方境界条件，右方――，上方――，下方――の順に判定データを作成します（ＧＣＢＤＲＹの項参照）．

カーソル中心座標によって一部が表示されないケース



ＧＣＢＤＲＹ

ＧＣＢＤＲＹ　　$Ｆ８８０　～　$Ｆ８ＥＢ

機　能	グラフィックカーソルの中で座標（Ｘ，Ｙ）をもとに，画面退避情報のヘッダ部分（図３・６・２参照）を作成します．	
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ（Ｘは保存）	
入力情報	Ｘレジスタ	画面情報退避場所の先頭アドレス
	ＧＣＸ	現在表示中のグラフィックカーソルの中心のＸ座標
	ＧＣＹ	現在表示中のグラフィックカーソルの中心のＹ座標
出力情報	画面退避情報のヘッダ部分［(Ｘ) 番地～(Ｘ＋５) 番地の６バイトに作成される］	
	ＧＣＢＩＴＭ　$Ｄ０ＢＡ	グラフィックカーソル用ビットマスク
ワークエリア	ＧＣＢＣＯＮ　$Ｄ０ＡＤ	各色画面データ退避用エリア１１バイトのうち実際に必要なバイト数が設定されるが，以後全く参照されない．
サブルーチン	ＧＡＤＤＲ	グラフィック座標→グラフィックアドレス生成［結果はＵレジスタ，同時にビットマスクがＢレジスタに格納される］
	ＬＳＲＤ３	Ｄレジスタを右に３ビットだけロジカルシフト

解　説　グラフィック座標ＧＣＸ，ＧＣＹに格納されているグラフィックカーソル中心座標を見て，カーソルが画面の四隅にかかっていないかどうか，またもし四隅にかかっているならば，どれ位かかっているかを判断してそれらの情報を（Ｘ＋２）～(Ｘ＋５）番地のヘッダ部分にまとめます．それと同時に，ＧＡＤＤＲによってＵレジスタに得られたグラフィックアドレス（ＶＲＡＭ相対アドレス）を（Ｘ）～(Ｘ＋１）番地に格納します．後述の，ＧＣＳＡＶＥルーチンを呼び出す前には，必ずこのＧＣＢＤＲＹルーチンで，あらかじめヘッダを作成しておかなければなりません．なお，Ｘレジスタの値は保存されますので，引き続いてＧＣＳＡＶＥルーチンを呼び出すことが可能なしくみになっています．というよりは，ＧＣＢＤＲＹルーチンは，ＧＣＳＡＶＥルーチンとペアで利用することを前提としてできている，といった方がよいかもしれません．

ＧＣＳＡＶＥ

ＧＣＳＡＶＥ　　＄Ｆ８ＥＣ　～　＄Ｆ９０Ｆ
ＧＣＳＡＶ１　　＄Ｆ９１０　～　＄Ｆ９４Ｃ

機　能　グラフィックカーソル表示位置にあたるＶＲＡＭの画面情報を，Ｘレジスタが示す番地以下のスタックエリアに退避させます．退避情報の形式は図３・６・２に示したとおりです．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ（Ｘは保存）

入力情報　Ｘレジスタ　画面情報退避場所の先頭アドレス
画面退避情報のヘッダ部分〔(Ｘ）～(Ｘ＋５）番地〕

出力情報　（Ｘ＋６）～(Ｘ＋３８）番地のスタックエリア上に，ＶＲＡＭから読み取った，各色画面の情報が格納される

ワークエリア　ＧＣＷＯＲＫ　＄Ｄ０ＢＣ～＄Ｄ０ＢＤ　グラフィックアドレス一時退避場所

サブルーチン　ＧＣＳＡＶ１　＄Ｆ９１０　※解説参照

解　説　この処理系において，実際にＶＲＡＭアクセスを行って画面情報を読み取るのは，ＧＣＳＡＶ１であり，ＧＣＳＡＶＥルーチンは，このＧＣＳＡＶ１をサブルーチンの形で呼び出すようになっておりますが，これは，青・赤・緑の各画面に対して，同一の処理を３回繰り返すからです．

なお，読取りの際には，ヘッダ部分（Ｘ）～(Ｘ＋５）番地の各境界条件を参照して，必要な情報だけを退避させるようにしています．

ＧＣＤＩＳＰ

ＧＣＤＩＳＰ	＄Ｆ９４Ｄ	～	＄Ｆ９８５
ＧＣＤＯＲ	＄Ｆ９８６	～	＄Ｆ９Ｄ１
ＧＣＤＡＮＤ	＄Ｆ９Ｄ２	～	＄ＦＡ２７

機　能　グラフィックカーソルの十字形を表示します．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ（Ｘは保存）

入力情報　Ｘレジスタ　　画面情報退避場所の先頭アドレス

画面退避情報のヘッダ部分〔(Ｘ)～(Ｘ＋５）番地〕

ＧＣＢＬＵＥ	＄Ｄ０Ｂ７	グラフィックカーソル表示色指定用のカラーエクステンドレジスタ〔点灯：＄ＦＦ，消灯：＄００〕
ＧＣＲＥＤ	＄Ｄ０Ｂ８	
ＧＣＧＲＥＥ	＄Ｄ０Ｂ９	
ＧＣＢＩＴＭ	＄Ｄ０ＢＡ	グラフィックカーソル用ビットマスク
ＧＣＣＵＲＳ	＄Ｄ０Ｂ０～＄Ｄ０Ｂ２	カーソルパターンレジスタ

ワークエリア

ＧＣＷＯＲ	＄Ｄ０ＢＢ	ビットマスクＧＣＢＩＴＭの反転
ＧＣＷＯＲＫ	＄Ｄ０ＢＣ～＄Ｄ０ＢＤ	グラフィックアドレス一時退避場所

サブルーチン

ＧＣＤＯＲ	＄Ｆ９８６	カーソル表示（点灯カラー用）
ＧＣＤＡＮＤ	＄Ｆ９Ｄ２	カーソル表示（消灯カラー用）

解　説　この処理系も，ＧＣＤＯＲおよびＧＣＤＡＮＤが，ＧＣＤＩＳＰのサブルーチンの形になっていて，構造化されております．青・赤・緑の各画面に関し，カーソル表示色を構成する原色にあたる画面に対しては，ＧＣＤＯＲを用いて表示を行い，カーソル表示色を構成しない原色にあたる画面に対しては，ＧＣＤＡＮＤを用いて表示を行います．

カーソルの表示は，上の縦棒／下の縦棒／左の横棒／中央／右の横棒の順番で行われますが，このうち上下の部分については，グラフィックカーソル用ビットマスクＧＣＢＩＴＭ（もしくはその反転）を，左／中央／右の部分については，カーソルパターンレジスタＧＣＣＵＲＳ（もしくはその反転）を，それぞれ利用して実行します．

ここでも，ヘッダ部分の各境界条件が参照され，画面の四隅を越えるカーソル部分については表示しないようになっています．

ＧＣＲＳＡＶ

ＧＣＲＳＡＶ　　$ＦＡ２８ ～ $ＦＡ４Ｂ
ＧＣＲＳＡ１　　$ＦＡ４Ｃ ～ $ＦＡ８９

機　能　ＧＣＳＡＶＥによって退避されたＶＲＡＭ画面情報を元どおりの位置に復帰させます.

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ（Ｘは保存）

入力情報　Ｘレジスタ
画面退避情報〔(Ｘ）～(Ｘ＋３８）番地〕

ワークエリア　ＧＣＷＯＲＫ　$Ｄ０ＢＣ～$Ｄ０ＢＤ　グラフィックアドレス一時退避場所

サブルーチン　ＧＣＲＳＡ１　$ＦＡ４Ｃ

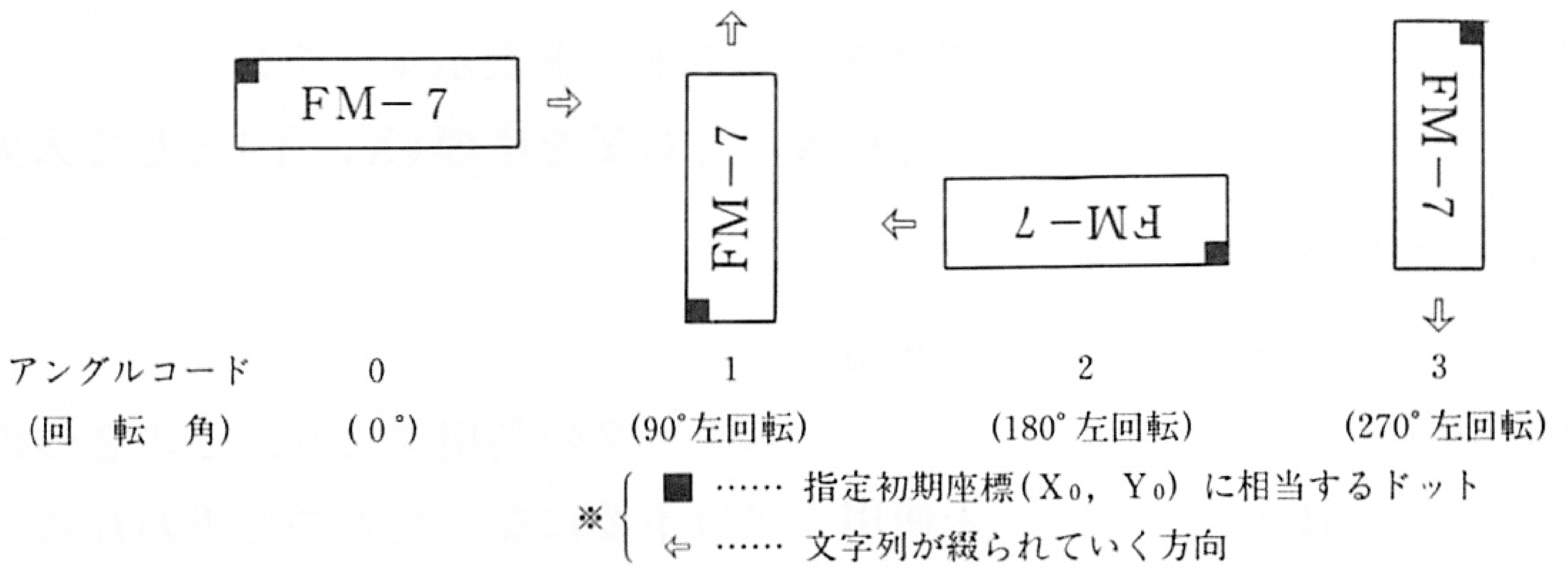
解　説　前述のＧＣＳＡＶＥルーチンと全く逆の働きをする処理系で，ＧＣＲＳＡＶとＧＣＲＳＡ１の関係は，ＧＣＳＡＶＥとＧＣＳＡＶ１の関係と同様です.

ユーザが，ＧＣＵＲＳ処理系と類似のプログラムを作成して，このＧＣＲＳＡＶを呼び出す場合には，必ず，後に退避させた情報から順に復帰させていくようにしなければならない点に注意して下さい．さもないと，カーソルの十字形に重なりがあった場合に，カーソルの一部が残ってしまう可能性があるからです．ＧＣＵＲＳメインループのエンディングルーチンでも，後のものから順に退避させていくようになっていますので，参考にして下さい.

3－6－2 SYMBOL $19 ($F103)

文字列を指定座標(X_0, Y_0)から，指定角度(0°，90°，180°，270°のいずれか)，指定した大きさで表示するコマンドです．この際文字を構成するドット以外の画面データは元のまま保存されます．それではアングルコード（角度指定）および指定初期座標（X_0, Y_0）と文字列が表示される形状との関係を以下に示します．

図3・6・4 SYMBOLコマンドによって表示される文字列の形状



共有RAM上のパラメータ

アドレス	記　号	内　容	値
D383	CL	カラーコード	0(黒)～7(白)
D384	F	ファンクションコード	0(PSET)～5(NOT)
D385	A	アングルコード (角度指定用コード)	0：ノーマル(0°) 1：90°左回転 2：180°左回転 3：270°左回転
D386	WM	文字横幅倍率(8ビット)	0～255($FF)
D387	HM	文字縦幅倍率(8ビット)	0～255($FF)
D388 ～ D389	IX(X_0)	表示開始位置のX座標 (16ビット)	0～639($027F) [グラフィック座標]
D38A ～ D38B	IY(Y_0)	表示開始位置のY座標 (16ビット)	0～199($00C7) [グラフィック座標]
D38C	N	文字列の文字数	1～80($50)
D38D ～ (D38C+N)	String	文字列	キャラクタコードの列

主なサブルーチン

$E383	GAXC	グラフィック座標（X，Y）のチェック［(X，Y）が（0，0）～(639，199）の範囲内にあるとき，キャリーセット］
$E3B7	CFADR	キャラクタコードからキャラクタフォントアドレスを生成［Aレジスタにキャラクタコード入力］
$E46B	GPDSIR	グラフィックドット表示用の初期化ルーチン［A：カラーコード，B：ファンクションコード］
$E48F	GPDSP	グラフィックドット表示ルーチン［.GX，.GYを座標(X，Y)として入力］

補助ワークエリア

$D0A0 ～$D0A1	WAX	未使用 ｝スタックの利用により，この2つのエリアは不要になったものと思われる.
$D0A2 ～$D0A3	HAX	未使用 ｝
$D0A4 ～$D0A5	WSV	文字列の横幅（W）方向のドットステップ［方向によって$0001／$FFFF(－1)］
$D0A6 ～$D0A7	HSV	文字列の高さ（H）方向のドットステップ［方向によって$0001／$FFFF(－1)］
$D0A8 ～$D0A9	WCSV	横幅(W)方向の倍率［方向によって，共有RAMのWMの値を16ビットに拡張した値，もしくは，その値の2の補数（NEG）の値が代入される］
$D0AA ～$D0AB	HCSV	高さ(H)方向の倍率［方向によって，共有RAMのWHの値を16ビットに拡張した値，もしくは，その値の2の補数（NEG）の値が代入される］
$D0AC	NOTF	NOTファンクション（フォント反転）フラグ［NOTのとき$FF／それ以外のとき$00］
$D0AD ～$D0AE	CPTR	文字列ポインタ［現在表示中のキャラクタ位置を指し示すポインタ，値は$D38D～$D3DC］
$D0AF ～$D0B0	WOP	未使用 ｝スタックの利用により，この2つのエリアは不要になったものと思われる.
$D0B1 ～$D0B2	HOP	未使用 ｝

＄Ｄ０Ｂ３	ＦＭＳＫ	現在表示中のフォントの位置を指し示すためのビットマスク
＄Ｄ０Ｂ４ ～＄Ｄ０Ｂ５	ＦＰＴＲ	フォントポインタ［テンポラリフォントエリア内の現在表示中のフォントの位置を指すポインタ］
＄Ｄ０Ｂ６	ＬＷ	文字の横幅方向（W方向）のループカウンタ［WMの値からダウンカウントされていく］
＄Ｄ０Ｂ７	ＬＨ	文字の高さ方向（H方向）のループカウンタ［HMの値からダウンカウントされていく］
＄Ｄ０Ｂ８ ～＄Ｄ０ＢＦ	ＳＹＭＦＮＴ	テンポラリフォントエリア［現在表示中の文字のキャラクタフォントを一時的に蓄えておくレジスタ］

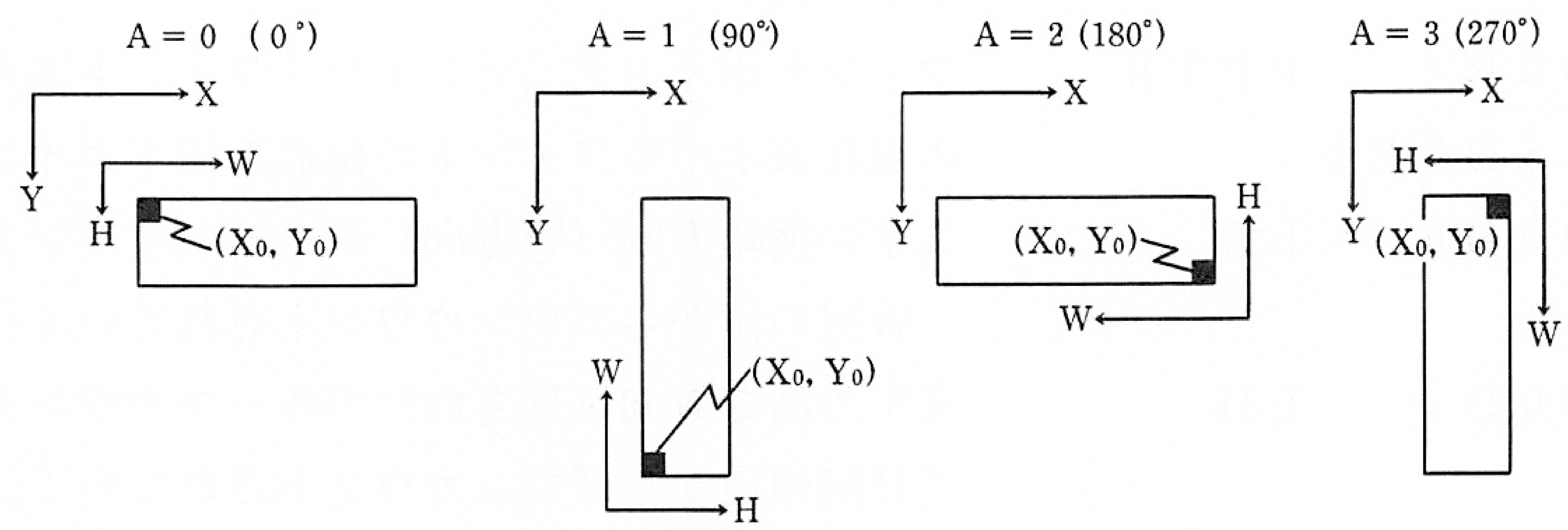
※共有RAM上のＩＸ，ＩＹ，Ｎの３つもポインタ・カウンタとして使用．

解　説　ＳＹＭＢＯＬ処理系は，ＧＡＸＣ，ＣＦＡＤＲ，ＧＰＤＳＩＲ，ＧＰＤＳＰの４つの外部汎用サブルーチンを呼び出すだけで，内部にサブルーチンすら持たないメインループのみで構成されている処理系ですが，だからといって，それは簡単な構造であることを意味しません．メインループだけで成り立っているとはいえ，ループは５重にネストしていますし，スタックをワークエリアとして利用し，しかも間接アドレッシングの媒体（仲介役）として，このスタックエリアを参照しているので，短いメインループの中に高級なテクニックが凝縮されているといった感じです．

ＳＹＭＢＯＬ処理系の最大のネックは，文字列の横幅方向Wおよび高さ方向Hからなるキャラクタフォント座標系（W，H）と，画面上のグラフィック座標系（X，Y）との間で座標変換を行う，という考え方を機軸にして表示を進めていく点です．そして，この座標変換を行う際に活躍する技法が，さきに述べたスタック上のワークエリアを媒介とする間接アドレスなのです．ユーザが指定したスタートポジションを，例えば(X_0, Y_0)とすると，座標変換の式は次ページのようになります．数学的には，このような変換のことをアフィン変換といいます．

さて，ＳＹＭＢＯＬコマンドによって描かれる文字列の表示順序は，アングルコードAがいくつであっても，キャラクタフォント座標系（W，H）で見れば，図３・６・５のようになります．この順序と方向をグラフィック座標系（X，Y）上で実現させるために，スタックエリアおよび補助ワークエリアを次のように初期設定します．

SYMBOL系における，座標変換の表式



$$\begin{cases} X = X_0 + W \\ Y = Y_0 + H \end{cases} \quad \begin{cases} X = X_0 + H \\ Y = Y_0 - W \end{cases} \quad \begin{cases} X = X_0 - W \\ Y = Y_0 - H \end{cases} \quad \begin{cases} X = X_0 - H \\ Y = Y_0 + W \end{cases}$$

ＳＹＭＢＯＬ処理系におけるワークエリアの初期設定

エリアの名前	ア ン グ ル コ ー ド（角 度）			
	A＝0（0°）	A＝1（90°）	A＝2（180°）	A＝3（270°）
S　W （S）～（S＋1）	＄D064 （.GCX）	＄D062 （.GCY）	＄D064 （.GCX）	＄D062 （.GCY）
S　H （S＋2）～（S＋3）	＄D062 （.GCY）	（.GCX）	（.GCY）	（.GCX）
S I W （S＋4）～（S＋5）	＄D388 （IX）	（IY）	（IX）	（IY）
S I H （S＋6）～（S＋7）	＄D38A （IY）	（IX）	（IY）	（IX）
W C S V ＄D0AA～＄D0A9	WM (横方向倍率)	－WM	－WM	WM
H C S V ＄D0AA～＄D0AB	HM (縦方向倍率)	HM	－HM	－HM
W S V ＄D0A4～SD0A5	＄0001 （＋1ドット）	＄FFFF （－1）	＄FFFF （－1）	＄0001 （＋1）
H S V ＄D0A6～＄D0A7	＄0001 （＋1ドット）	＄0001 （＋1）	＄FFFF （－1）	＄FFFF （－1）

図3・6・5　SYMBOL文字列の表示順序

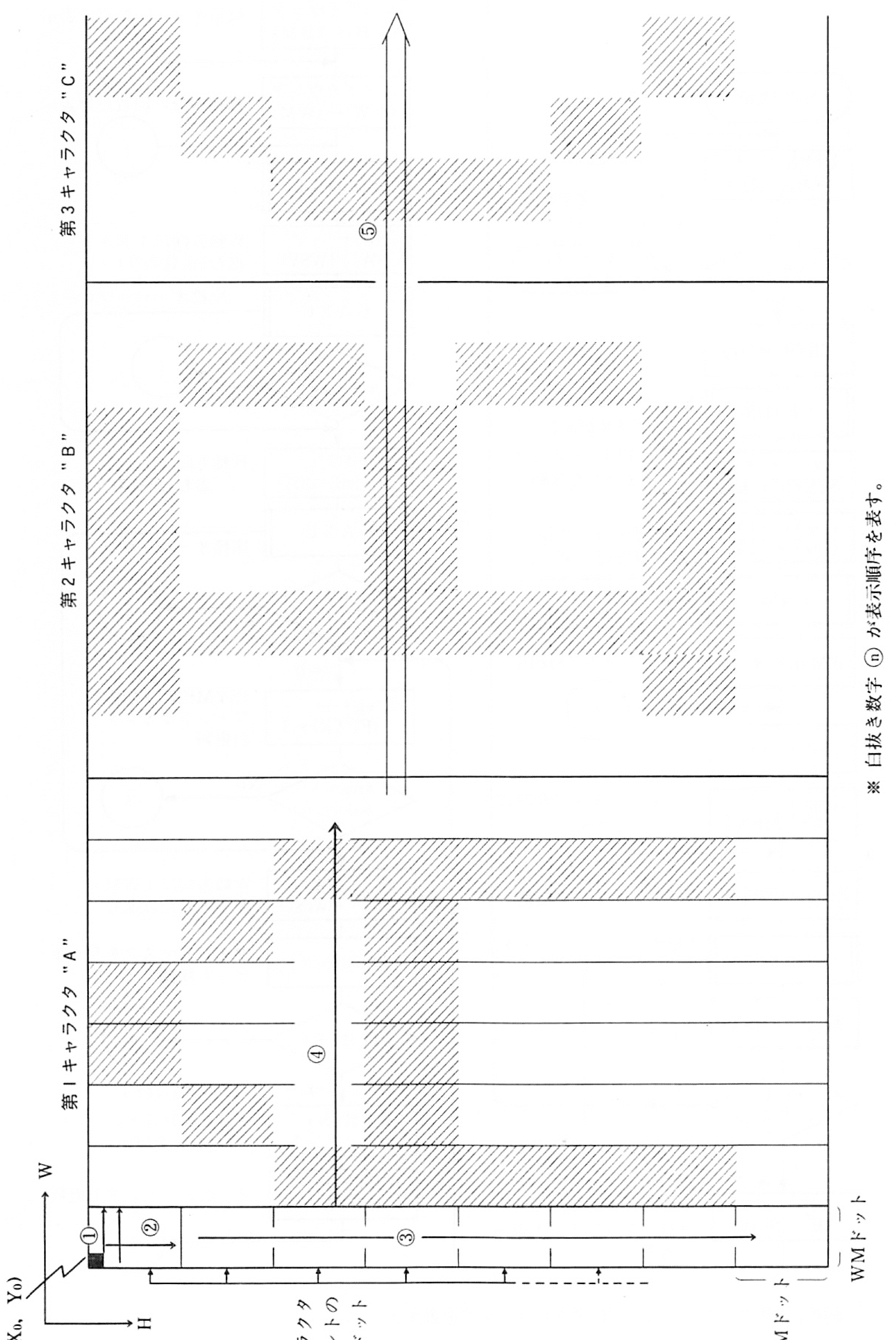
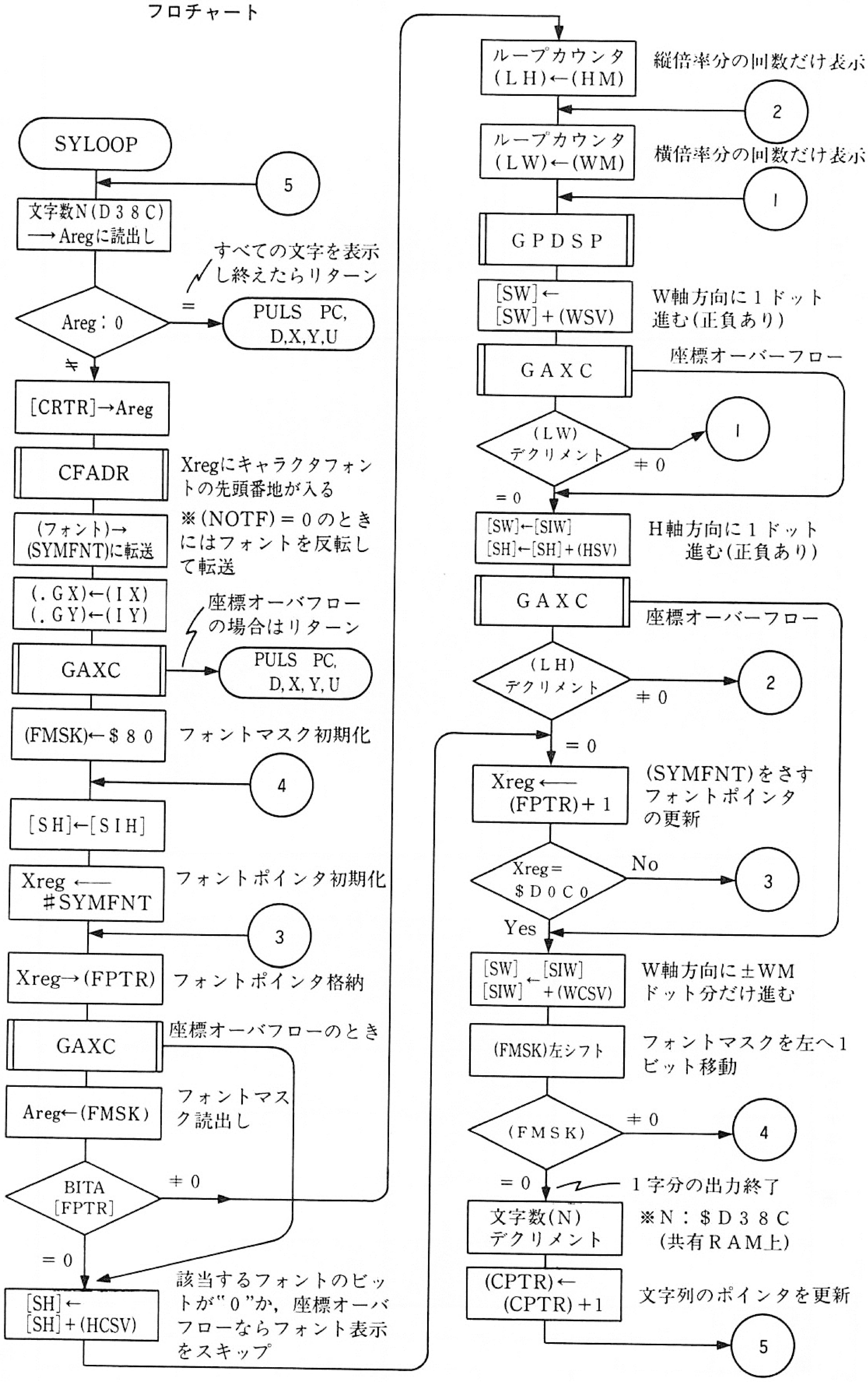


図３・６・６　SYMBOLメインループのフロチャート



※図中の角かっこ[　]は，間接アドレッシングを表す。

それでは，ＳＹＭＢＯＬ処理系のアルゴリズムについて解説します．

ＳＹＭＢＯＬ処理系に入ると，まず，カラーコードおよびファンクションコードをＤレジスタに読み出して，ドット表示用初期設定ルーチンＧＰＤＳＩＲを呼び出します．ただし，ファンクションコードＦ＝５の場合には，フォント反転フラグNOTＦを立てて，Ｆ＝０（ＰＳＥＴ）としてＧＰＤＳＩＲを呼び出します．この後，上表のとおりにワークエリアを初期設定してから，処理系のメインループＳＹＬＯＯＰに入ります．ＳＹＬＯＯＰのアルゴリズムを図３・６・６に掲げます．ここで，①～⑤のループ番号は，図３・６・５において表示の順序を表す番号①～⑤と対応しております．また，注意しなければならない点は，共有ＲＡＭ上のパラメータ：ＩＸ，ＩＹ，Ｎの値も，ポインタ・カウンタとして更新されていくという点です．つまり，共有ＲＡＭもワークエリアとして利用しているわけで，ＳＹＭＢＯＬ実行中にメイン側でＨＡＬＴをかけてこれらのパラメータの値を破壊すると，とんでもない動作を行う可能性があります．このことは，裏を返せばメイン側から適当な間隔でＨＡＬＴをかけて共有ＲＡＭのパラメータを読み込むことによって，処理の進行状況をメイン側で監視することも可能であるということになります．

3－6－3　Get Block 1　$1B　($F286)
Get Block 2　$1D　($F3B6)

Get Block 1では，対角線座標（X1，Y1）－（X2，Y2）で示される枠内の，指定された色のドットを1，その他の色のドットを0として，画面のビットイメージを読み取ります．色の指定は最大8色まで可能で，指定されたいくつかのカラーコードのうちどれか一つでも持っていれば，そのドットのイメージは“1”となります．例えば，赤・紫・黄の3色を指定した場合，赤または紫または黄のカラーコードを持つドットのイメージは“1”に，青・緑・水色・白・黒のカラーコードを持つドットのイメージは“0”となります．

一方，Get Block 2では枠内の3原色（青・赤・緑）のデータを読み取ります．

共有RAM上のパラメータ

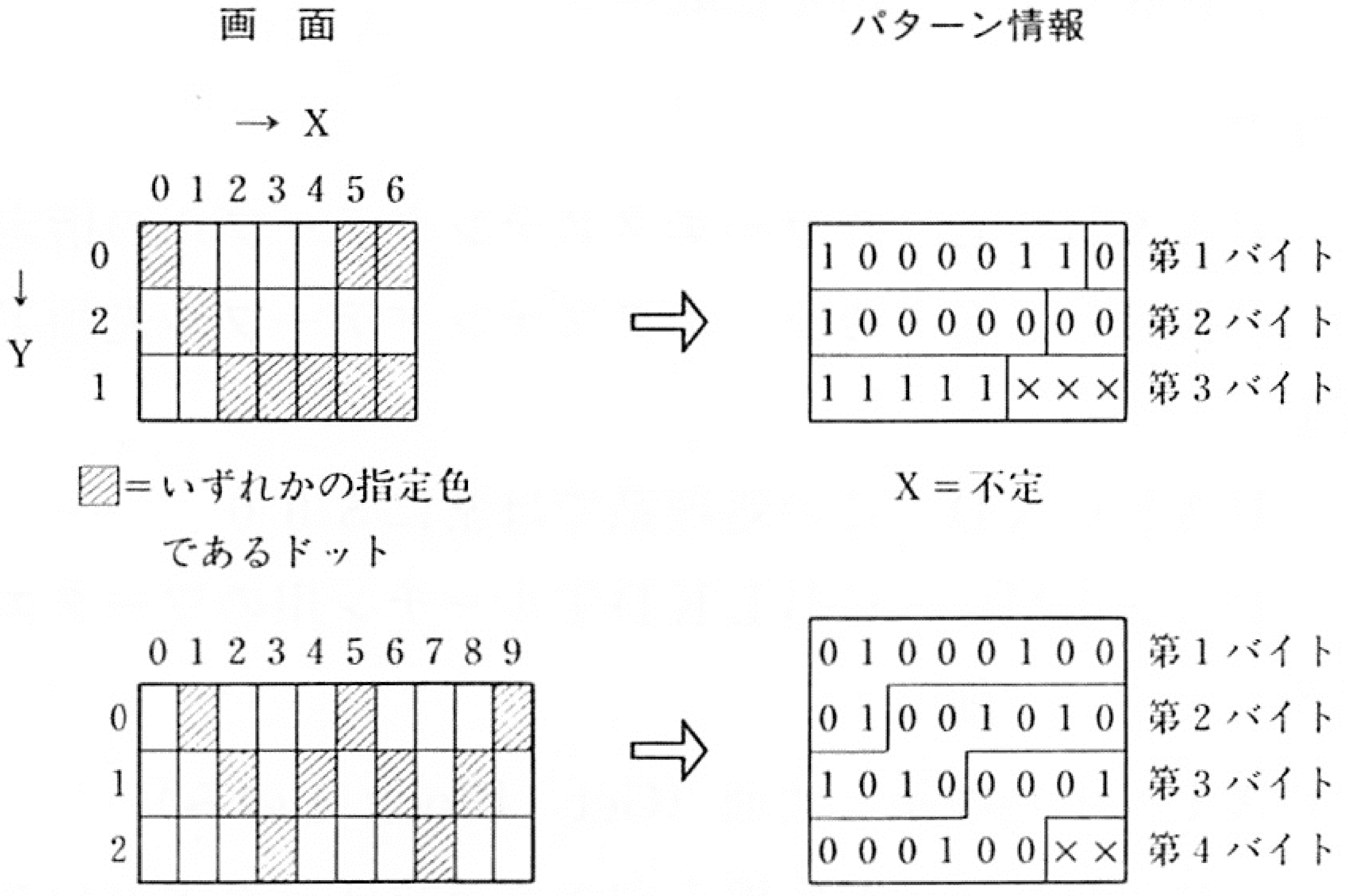
アドレス	記　号	内　容	値
D383 ～ D384	X1	X1座標（16ビット）	0～639（グラフィック座標）
D385 ～ D386	Y1	Y1座標（16ビット）	0～199（グラフィック座標）
D387 ～ D388	X2	X2座標（16ビット）※X1≦X2である必要はない	0～639
D389 ～ D38A	Y2	Y2座標（16ビォト）※Y1≦Y2である必要はない	0～199
D38B	N	指定色の数	1～8
D38C	CL1（CLn）	指定色カラーコード1～N　（Nバイト）	0～7

※上表は，Get Block 1の場合．Get Block 2では，$D38B以下は不要となる（X1～Y2までよい）

共有RAM上の復帰情報

アドレス	記　号	内　容	値
D383	Byte	今回転送するバイト数	1～124
D384 ～ D383＋Byte	Pattern	画面データ（ビットイメージの集合体）	パターン情報

Get Block 1の場合，画面のドット配置と読み込まれるパターン情報との対応関係は次のようになります．



Get Block 2では，画面のパターン情報は，青（B），赤（R），緑（G）の各原色ごとに読み取られ，この順番で，共有ＲＡＭに転送されます．各原色単位での，画面ドット配置と転送されるパターン情報との対応関係は，上記のGet Block 1の場合とほぼ同じです．

主なサブルーチン

処理系内部

＄Ｆ２ＣＢ	ＧＢＩＮＴ	初期設定ルーチン
＄Ｆ２ＥＤ	ＣＬＥＸＴ	カラーエクステンドテーブル作成ルーチン
＄Ｆ３０８	ＧＢＰＵＴ	共有ＲＡＭ出力／ＣＯＮＴコマンド待ち
＄Ｆ３２５	ＧＢＰＬＡ	行データ変形（ビットシフト）／出力
＄Ｆ３ＦＣ	ＧＢＯＩＮＴ	出力準備（初期化）ルーチン
＄Ｆ３ＣＦ	ＧＢＬＫ２Ｓ	Get Block 2の実行サブルーチン

処理系外部

＄Ｅ０Ｆ５	ＣＯＮＴＯ	ＣＯＮＴコマンド待ち（サブ⇒メイン出力用）
＄Ｅ３Ｆ２	ＧＸＯＦＦ	Ｘ座標方向のグラフィックアドレス相対値生成，整数除算ルーチンとしても利用 (Dreg￥8 → Xreg, Dreg mod 8→ Breg)
＄Ｅ４０２	ＢＯＸＩＮＶ	対角線座標読取り（エラー付き）
＄Ｅ４４Ａ	ＭＯＶＥ	ブロック転送（Get Block 2のみ使用）
＄ＦＣ８６	ＧＢＬＫＤＴ	行データ読込み（ＰＡＩＮＴ処理系内のサブルー

チンＰＡＢＧＥＴと同一，エントリのみが異る)

エラー

＄Ｅ０Ｅ４	ＥＲＲ６７	指定色数エラー

補助ワークエリア

＄Ｄ０ＡＦ	ＰＡＮＢＣＬ	カラーエクステンドテーブルの指定色の数
＄Ｄ０Ｂ０ ～＄Ｄ０Ｂ１	ＰＡＢＣＬＥ	カラーエクステンドテーブル先頭アドレス
＄Ｄ０Ｂ３	ＰＡＳＣＮＤ	この処理系では常に＄００
＄Ｄ０Ｂ４ ～＄Ｄ０Ｂ８	ＰＡＳＩＭＧ 他	ＧＢＬＫＤＴルーチン用のワークエリア

※以上は，ＰＡＩＮＴ処理系と共通（Get Block 1のみ）

＄Ｄ０Ｂ９ ～＄Ｄ０ＢＡ	ＧＢＡＤＸ	Ｘ座標方向のグラフィックアドレス相対値 (座標Ｘ１に対するもの，Ｘ１ ¥ ８の値)
＄Ｄ０ＢＢ	ＧＢＤＰＸ	座標Ｘ１に対するビットナンバ(Ｘ１ mod ８の値)
＄Ｄ０ＢＣ	ＧＢＮＢＸ	Ｘ座標方向のバイト数（行データのバイト数に相当，Ｘ２ ¥ ８ － Ｘ１ ¥ ８の値が入る)
＄Ｄ０ＢＤ ～＄Ｄ０ＢＥ	ＧＢＮＸ	Ｘ座標方向のドット数（行データのドット数に相当，Ｘ２－Ｘ１＋１の値が入る)
＄Ｄ０ＢＦ	ＧＢＮＹ	Ｙ座標方向のドット数（Ｙ２－Ｙ１＋１の値)
＄Ｄ０Ｃ０ ＄Ｄ０Ｃ１	ＧＢＩ ＧＢＪ	制御変数"Ｉ"，"Ｊ"（ＦＯＲ～ＮＥＸＴループの"Ｉ＝１　ＴＯ　５"の"Ｉ"に相当する)
＄Ｄ０Ｃ２ ～＄Ｄ０Ｃ３	ＧＢＰＩＸ	出力バッファ（共有ＲＡＭ上）ポインタ
＄Ｄ０Ｃ４	ＧＢＩＸ	行データの剰余データのドット数(剰余ドット数) ※ＧＢＰＬＡを続けて呼び出す際の引継ぎ用数値
＄Ｄ０Ｃ５ ～＄Ｄ０Ｃ６	ＧＢＴＭＩＸ	ＶＲＡＭアドレス一時退避用 (Get Block 2のみ使用)
＄Ｄ０Ｃ７ ～＄Ｄ０Ｃ８	ＧＢＤＩＸ	枠内の座標（Ｘ１，Ｙ)〔Ｙ１≦Ｙ≦Ｙ２〕に対するグラフィックアドレスをさすポインタ (行データの先頭アドレスをさす)
＄Ｄ０Ｃ９	ＧＢＫ	制御変数"Ｋ"(Get Block 2のみ使用)
＄Ｄ０ＣＡ	ＧＢＳＶ	行データ変形の際のシフトドット数
＄Ｄ０ＣＢ	ＧＢＣＬ	カラーエクステンドテーブル

～＄Ｄ０Ｅ２　　　　　　　　（3バイト×8色＝24バイト）
＄Ｄ０Ｅ３　ＧＢＤＡＴＡ　行データ格納場所（＄Ｄ０Ｅ３は剰余データの格
～＄Ｄ１３５　　　　　　　納場所であり，実際の行データは＄Ｄ０Ｅ４から）

解　説　この処理系は，対角線座標の枠で示された画面上のｘ×ｙの形の配列を，８×ｎの形の配列に並び換えるところにポイントがあります．ｘが常に8の倍数であれば問題ないのですが，実際にはそうでないことの方が多く，ビット操作を駆使して並べ換えを進めなければなりません．

具体例で見てみます．図３・６・７のように，（Ｘ１，Ｙ１）＝（３，５），（Ｘ２，Ｙ２）＝（２０，８）で指定されるような，１８×４の枠内のデータを読み取る場合について考えてみます．

まず，第１回目の転送で，Ｙ＝５に相当する部分のＶＲＡＭ画面データを，ＧＢＤＡＴＡエリアの＄Ｄ０Ｅ５番地以下に読み込みます．このような1ライン分の転送データを行データと呼ぶことにします．Ｘ１＝３，Ｘ２＝２０ですから，グラフィックアドレスにして＄０１９０～＄０１９２番地の3バイト分の転送データが行データとなります．

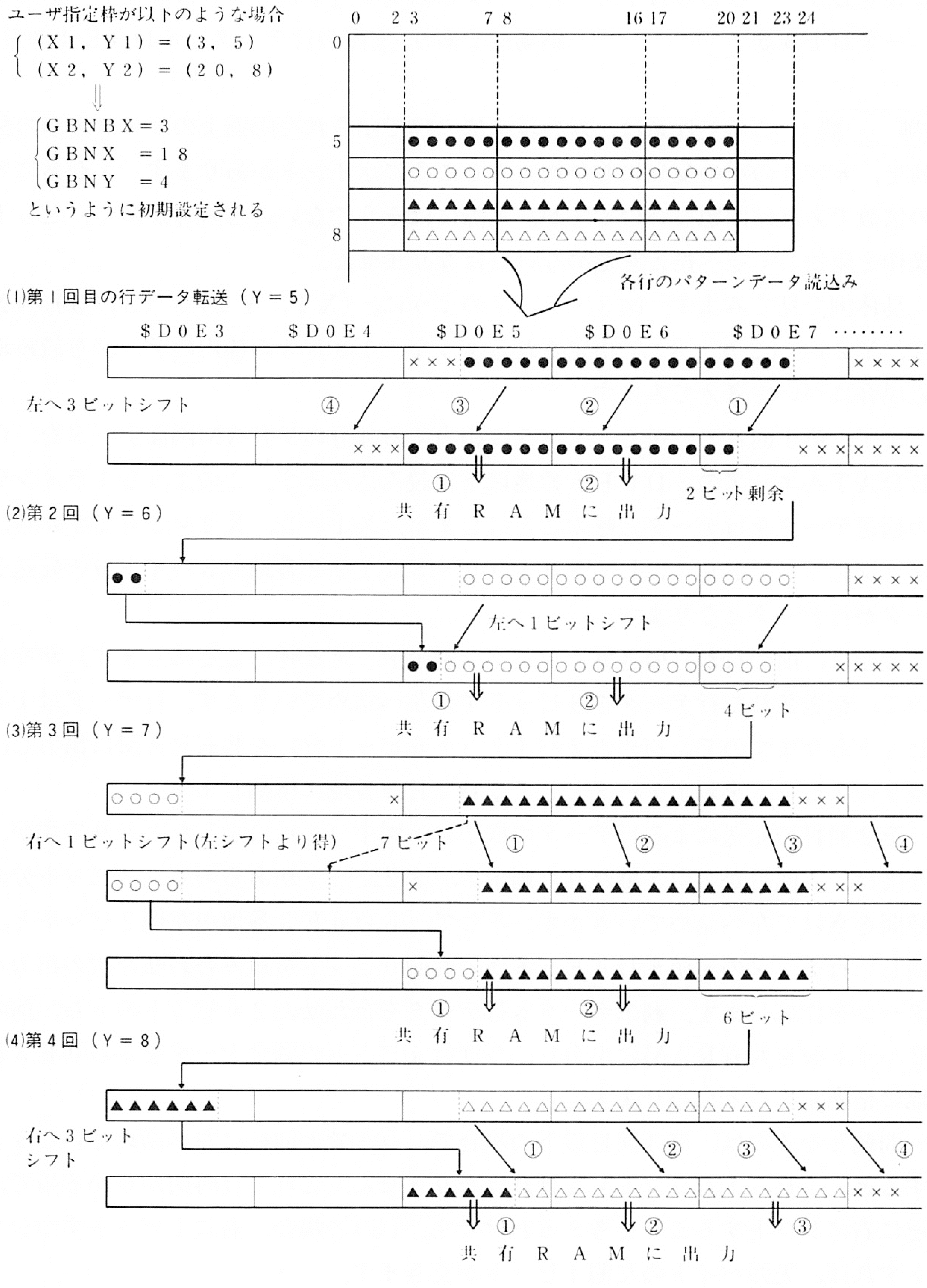
最初は，前の行から引き継ぐデータ（剰余データと呼ぶことにします）がないので，転送された行データを3ビットずつ左へ詰めていきます．行データは１８ビットありますので，初めの2バイト（１６ビット分）を共有ＲＡＭに出力し，残りの2ビットを剰余データとして＄Ｄ０Ｅ３番地に格納します．

第2回目の転送による行データの様子は，図中（２）に示したとおりですが，今度は，１回目から引き継がれた剰余データ2ビットがあるので，２ビット分の隙間を空けて左へ詰めていきます．そして，＄Ｄ０Ｅ３番地の左側2ビットと，新しい行データの先頭＄Ｄ０Ｅ５番地の右側6ビットを組み合わせた次の出力パターンを作成します．剰余データと行データを合わせた２０ビットのうち，前の2バイト分を共有ＲＡＭに出力し，今度は4ビットの剰余データを＄Ｄ０Ｅ３番地に格納することになります．

問題となるのは，第３回目以下の場合で，今までと同様に左へ詰めていこうとすると，７ビットもシフトしなければなりません．これでは時間がかかるので，逆に右にシフトすることを考えるわけです．（３）の場合，右に１ビットずつシフトすれば，先頭バイトの左側4ビットが空きます．

第4回目（４）では，剰余データと行データが合わせて２４ビットとなり，ちょうど3バイト分なので，これらすべてを出力して処理を終えます．もし仮に，最後の行データ転送後に剰余データが残っている場合には，この剰余データを出

図３・６・７　Get Blockにおける行データの読込み・変形と出力



力してから処理を終了させます．

さて，各回の行データ転送後のデータ変形を行うのが，ＧＢＰＬＡというラベル名のついたサブルーチンです．このルーチンでは，＄Ｄ０Ｅ５番地以下に格納された行データを，どちらの方向に何ビットずつシフトしたらよいかという判断をはじめ，シフト実行前の行の剰余データとの組合せ，新しい剰余データの格納，共有ＲＡＭへのデータ出力まで，Get Block の中心的な処理のほとんどが実行されます．データのシフトの方向およびビット数は，ワークエリアＧＢＳＶに計算されますが，この値は，下表のようになります．ただしＧＢＩＸは，前の行から引き継いだ剰余データのビット数ＧＢＤＰＸは，先頭データが何ビット目から始まるのかを表すビットナンバであり，表中の符号は正が右シフトの方向を，負が左シフトの方向を，それぞれ表します．

ＧＢＰＬＡルーチンにおけるＧＢＳＶ（シフトドット数）の計算

ＧＰＤＰＸの値（＄Ｄ０Ｅ４番地以下に格納されている行データの先頭データが何ビット目から始まるかを表すビットナンバ） ＼ ＧＢＩＸの値（剰余データのビット数）	0	1	2	3	4	5	6	7
0	0	1	2	3	4	−3	−2	−1
1	−1	0	1	2	3	4	−3	−2
2	−2	−1	0	1	2	3	4	−3
3	−3	−2	−1	0	1	2	3	4
4	−4	−3	−2	−1	0	1	2	3
5	3	−4	−3	−2	−1	0	1	2
6	2	3	−4	−3	−2	−1	0	1
7	1	2	3	−4	−3	−2	−1	0

この枠内に入るケースでは，シフト後のデータの先頭が＄Ｄ０Ｅ４番地となる。（右上の枠：GPDPX 0のGBIX 5〜7，1の6〜7，2の7）

この枠内に入るケースでは，シフト後のデータの先頭が＄Ｄ０Ｅ６番地となる．（左下の枠：GPDPX 5のGBIX 0，6の0〜1，7の0〜2）

上下の枠外のケースでは，シフト後のデータの先頭は＄Ｄ０Ｅ５番地となる。

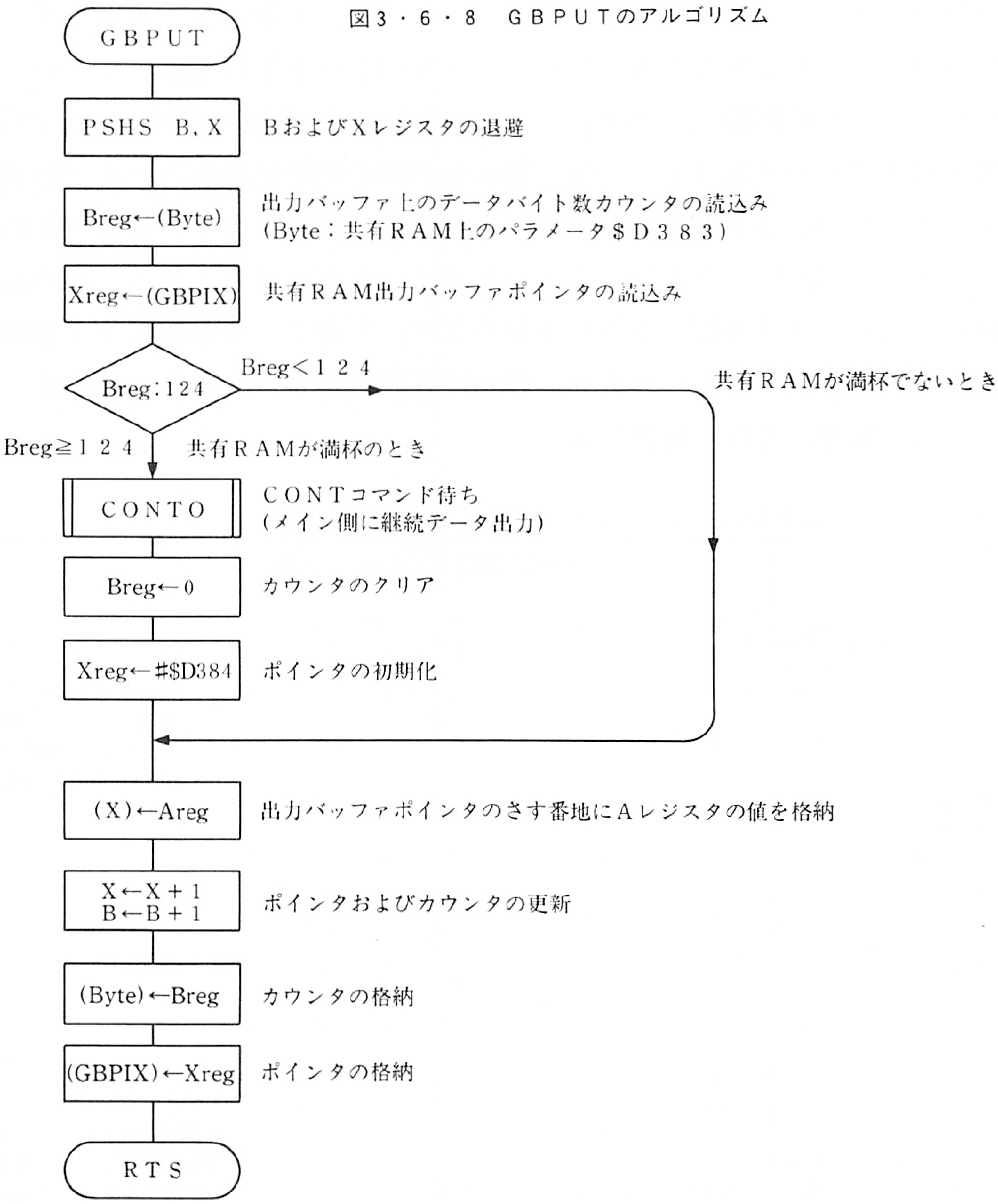
一方，共有ＲＡＭへの出力バイト数および新しい剰余データのビット数ＧＢＩＸは，次の計算式に従って求められます．

（共有ＲＡＭ出力バイト数）＝（（前のＧＢＩＸの値）＋ＧＢＮＸ）　¥　8
（新しいＧＢＩＸの値）　　＝（（前のＧＢＩＸの値）＋ＧＢＮＸ）mod　8

なお，共有ＲＡＭへの出力にはＧＢＰＵＴルーチンが用いられておりますが，このアルゴリズムを図３・６・８にあげておきます．共有ＲＡＭが一杯になった時点でＣＯＮＴＯを呼び出すことにより，メイン側と同期をとっている点に注目し

て下さい．

図３・６・８　ＧＢＰＵＴのアルゴリズム



それから，話の順序が逆になりましたが，各行の行データの転送には，Get Block 1ではＰＡＩＮＴ処理系内サブルーチンＧＢＬＫＤＴが，またGet Block 2では，ブロック転送ルーチンＭＯＶＥが，それぞれ用いられております．このうち，ＧＢＬＫＤＴはＰＡＢＧＥＴと同一のサブルーチンであり，エントリと入力情報の指定の仕方のみが異るだけですので，ＰＡＩＮＴコマンドの解説（３－５－４節）も合わせて御覧下さい．

ＧＢＩＮＴ

ＧＢＩＮＴ　　＄Ｆ２ＣＢ　～　＄Ｆ２ＥＣ

機　能	共有ＲＡＭ上のパラメータを読み取って，Get Block 処理用の初期設定を行います．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報	共有ＲＡＭ上のパラメータ：Ｘ１，Ｙ１，Ｘ２，Ｙ２ （＄Ｄ３８３　～　＄Ｄ３８Ａ）
出力情報	．ＢＯＸＸ１　＄Ｄ０６９～＄Ｄ０６Ａ　対角線Ｘ座標(小さい方) ．ＢＯＸＹ１　＄Ｄ０６Ｂ～＄Ｄ０６Ｃ　対角線Ｙ座標(小さい方) ．ＢＯＸＸ２　＄Ｄ０６Ｄ～＄Ｄ０６Ｅ　対角線Ｘ座標(大きい方) ．ＢＯＸＹ２　＄Ｄ０６Ｆ～＄Ｄ０６Ｅ　対角線Ｙ座標(大きい方) ＧＢＡＤＸ　＄Ｄ０Ｂ９～＄Ｄ０ＢＡ　Ｘ１　￥　８ ＧＢＤＰＸ　＄Ｄ０ＢＢ　ビットナンバ：Ｘ１ mod ８ ＧＢＮＢＸ　＄Ｄ０ＢＣ　横のバイト数：Ｘ２￥８－Ｘ１￥８＋１ ＧＢＮＸ　＄Ｄ０ＢＤ～＄Ｄ０ＢＥ　横（Ｘ）のドット数：Ｘ２－Ｘ１＋１ ＧＢＮＹ　＄Ｄ０ＢＦ　縦のドット数：Ｙ２－Ｙ１＋１ ＧＢＩＸ　＄Ｄ０Ｃ４　剰余ドット数＝０に設定
サブルーチン	ＢＯＸＩＮＶ　＄Ｅ４０２　対角線座標の読取り・並べ換え ＧＸＯＦＦ　＄Ｅ３Ｆ２　整数除算：Dreg￥８

解　説	共有ＲＡＭの対角線座標の読取りはＢＯＸＩＮＶルーチンで行ってくれるので，それらのデータを加工して，Get Block 処理用の初期設定を行います．設定の結果は出力情報にあげたとおりです．

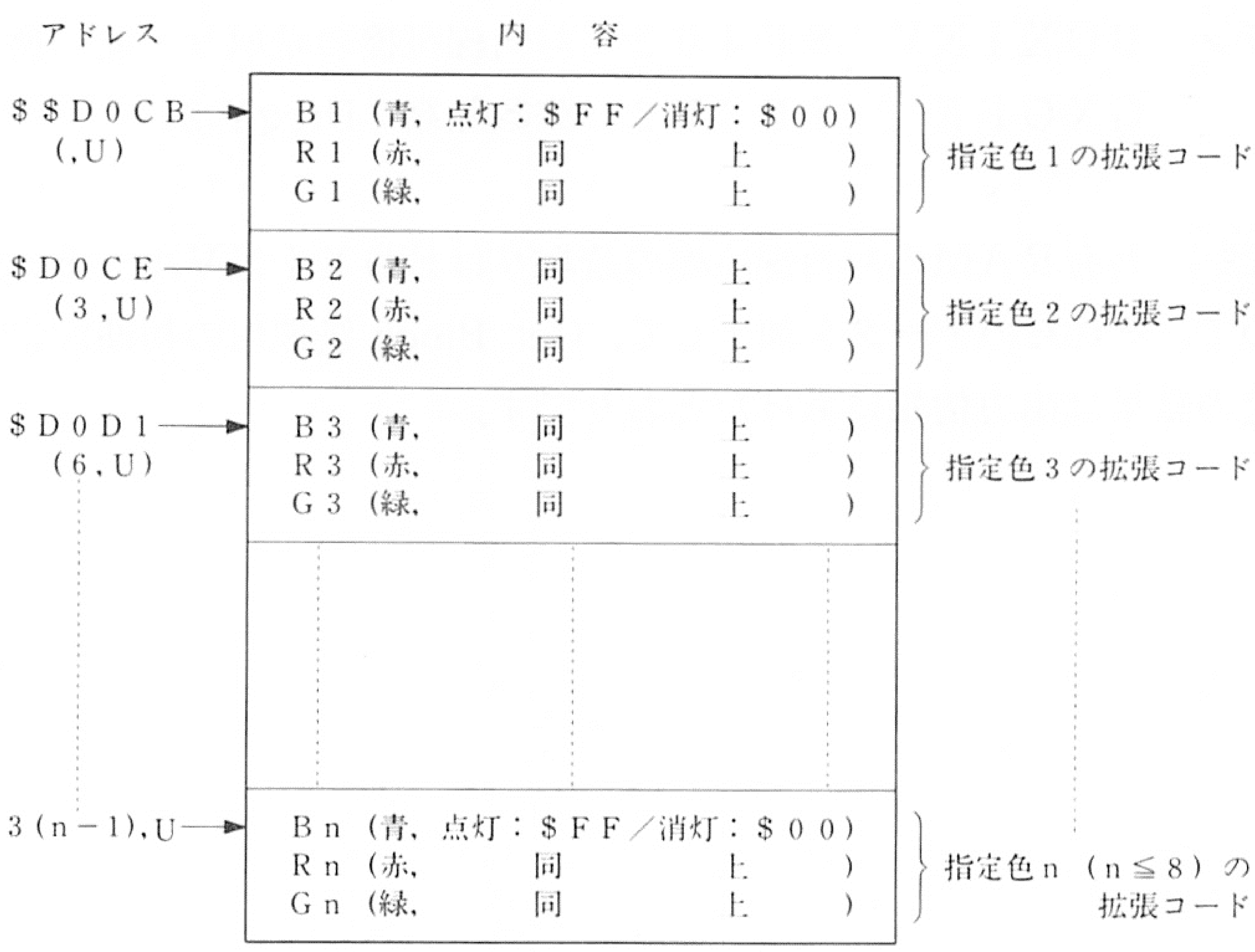
CLEXT

CLEXT　　$F2ED ～ $F300
GBSCL　　$F301 ～ $F307

機　能	GBLKDTルーチン用のカラーエクステンドテーブルを作成．
レジスタ	A，B，X，U
入力情報	Bレジスタ　指定色カラーコードの数（$D38Bの内容） Xレジスタ　指定色カラーコードテーブル先頭（#$D38C） Uレジスタ　カラーエクステンドテーブル先頭（#GBCL）
出力情報	(U)～(U+B＊3－1) 番地にカラーエクステンドテーブル PANBCL　$D0AF　指定色の数 PABCLE　$D0B0～$D0B1　テーブル先頭
サブルーチン	GBSCL　$F301

解　説　CLEXTとGBSCLの関係は，グラフィックカーソルで出てきたGCSAVEとGCSAV1の関係とよく似ております（3－6－1節参照）．作成されるカラーエクステンドテーブルは次のとおりです．

カラーエクステンドテーブルGBCLの様子



ＧＢＯＩＮＴ

ＧＢＯＩＮＴ　　　＄Ｆ３ＦＣ　～　＄Ｆ４１１

機　能	Get Block 系の出力の初期設定を行います．		
レジスタ	Ａ，Ｂ，Ｘ		
入力情報	．ＢＯＸＹ１	＄Ｄ０６Ｂ～＄Ｄ０６Ｃ	座標Ｙ１の値
	ＧＢＡＤＸ	＄Ｄ０Ｂ９～＄Ｄ０ＢＡ	Ｘ１　￥　８の値
出力情報	ＧＢＰＩＸ	＄Ｄ０Ｃ２～＄Ｄ０Ｃ３	＄Ｄ３８４に設定
	ＧＢＩＸ	＄Ｄ０Ｃ４	剰余ドット数＝０に設定
	ＧＢＤＩＸ	＄Ｄ０Ｃ７～＄Ｄ０Ｃ８	座標（Ｘ１，Ｙ１）に対応するグラフィックアドレスが格納される．

解　説　出力データを格納する共有ＲＡＭをさす，結果格納用ポインタＧＢＰＩＸに＄Ｄ３８４を代入し，蓄積された結果データの個数をかぞえるカウンタByte（＄Ｄ３８３）をクリアします．この他，データ先頭ドットに対応するグラフィックアドレスの値を計算してＧＢＤＩＸに格納します．

ＧＢＰＬＡ

ＧＢＰＬＡ　　　＄Ｆ３２５　～　＄Ｆ３Ｂ５

機　能	行データの変形・出力を行います．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	ＧＢＤＡＴＡ	＄Ｄ０Ｅ３～	剰余データ（＄Ｄ０Ｅ３）および行データ（＄Ｄ３Ｅ５～）
	ＧＢＤＰＸ	＄Ｄ０ＢＢ	行データ先頭のビットナンバ(０～７)
	ＧＢＮＢＸ	＄Ｄ０ＢＣ	行データのバイト数（１～８０）
	ＧＢＮＸ	＄Ｄ０ＢＤ～	行データのビット数（１～６４０）
	ＧＢＩＸ	＄Ｄ０Ｃ４	剰余データのビット数（０～７）
出力情報	共有ＲＡＭ上にコマンド処理の結果		
	ＧＢＤＡＴＡ	＄Ｄ０Ｅ３	新しい剰余データ
	ＧＢＩＸ	＄Ｄ０Ｃ４	新しい剰余データのビット数
ワークエリア	ＧＢＩ	＄Ｄ０Ｃ０	ループカウンタ
	ＧＢＪ	＄Ｄ０Ｃ１	ループカウンタ
	ＧＢＳＶ	＄Ｄ０ＣＡ	シフトドット数

サブルーチン　GXOFF　$E3F2　除算（¥8，mod 8）ルーチン
GBPUT　$F308　出力ルーチン

解　説　先程も触れたとおり，行データをまとめてシフトして，剰余データと組み合わせ，共有RAMに出力します．剰余データの左側を取り出して行データ先頭の右側と合成して8ビットデータを生成する際，マスクが必要となりますが，このマスクの作り方に面白い技法が用いられております．剰余ビット数GBIXから，ビットマスクテーブルBAS（$E392～$E399）を参照してBレジスタにビットマスクを求め，これを2倍して1を引くことにより，右側データ用マスク（行データ側）を作成しています．左側マスクはこれを反転することにより求められます．

GBPUT

GBPUT　$F308 ～ $F324

機　能　共有RAMにデータを出力します．この際共有RAMが一杯になっていた場合には，CONTOを呼び出してCONTコマンド待ちに入ることにより，メイン側との同期をとります．

レジスタ　A，B，X（すべて保存される）

入力情報　Aレジスタ　出力データ

ワークエリア　GBPIX　$D0C2～$D0C3　共有RAM出力バッファポインタ
共有RAM転送バイト数 Byte　$D383　バイトカウンタ

サブルーチン　CONTO　$E0F5　CONTコマンド待ち（出力用）

解　説　アルゴリズムは，前述の図3・6・8のとおりで，バイトカウンタをテストして，共有RAMが満杯でなければ，GBPIXで示されるところの共有RAMバッファにデータを格納して，バッファポインタおよびバイトカウンタを1つずつインクリメントして更新します．満杯になっていたら，CONTOルーチンを呼び出してCONTコマンド待ちを行い，カウンタ Byte（$D383）およびポインタGBPIX（$D0C2～$D0C3）の初期化を行ってから，出力データ（Aレジスタ）を格納します．

ＧＢＬＫＤＴ

ＧＢＬＫＤＴ　　$ＦＣ８６ ～ $ＦＣＣ１

機　能	カラーエクステンドテーブルにおける指定色を参照して，Ｕレジスタ内のグラフィックアドレスの位置の画面データに対するビットイメージを作成します．Get Block 1における行データというのは，このビットイメージの集合体にほかなりません．		
レジスタ	Ａ，Ｂ，Ｘ，Ｕ（Ｕは保存）		
入力情報	Ｕレジスタ	ビットイメージを求めようとしているグラフィックアドレス	
	ＰＡＮＢＣＬ	$Ｄ０ＡＦ	指定色の数
	ＰＡＢＣＬＥ	$Ｄ０Ｂ０～$Ｄ０Ｂ１	テーブル先頭アドレス
出力情報	ＰＡＳＩＭＧ	$Ｄ０Ｂ４	求められたビットイメージ
	Ｂレジスタ		求められたビットイメージ
ワークエリア	ＰＡＴＥＭＰ	$Ｄ０Ｂ５	
	ＰＡＶＢ	$Ｄ０Ｂ６	
	ＰＡＶＲ	$Ｄ０Ｂ７	
	ＰＡＶＧ	$Ｄ０Ｂ８	

解　説　先程も触れましたが，このルーチンは，ＰＡＩＮＴ処理系サブルーチンと，エントリ・入力情報のみが異る同一のルーチンです．カラーエクステンドテーブルに，３バイトずつに拡張された８色までの指定色のうち，どれか一つでも画面のカラーと一致した場合には，そのドットに対応するイメージは“１”とします．いくつかある指定色に対してＯＲをとっていくことになります．詳しくは，ＰＡＢＧＥＴの解説（３－５－４節）を参照して下さい．

ＧＢＬＫ２Ｓ

ＧＢＬＫ２Ｓ　　$ＦＣ３Ｆ 〜 $Ｆ３ＦＢ

機　能	Get Block 2のサブルーチンで，青・赤・緑の各画面ごとのパターン情報を作成・出力していきます．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報	Ｘレジスタ　現在処理中のＶＲＡＭアドレス ＧＢＮＹ　$Ｄ０ＢＦ　Ｙ方向のドット数（Ｙ２－Ｙ１＋１） ＧＢＮＢＸ　$Ｄ０ＢＣ　Ｘ方向のバイト数 など
出力情報	ＧＢＩＸ　$Ｄ０Ｃ４　０にクリアされる
ワークエリア	ＧＢＴＭＩＸ　$Ｄ０Ｃ５〜$Ｄ０Ｃ６　ＶＲＡＭアドレス一時退避用 ＧＢＫ　$Ｄ０Ｃ９　ループカウンタ
サブルーチン	ＭＯＶＥ　$Ｅ４４Ａ　ブロック転送（Ｘ→Ｙ，Ｂバイト） ＧＢＰＬＡ　$Ｆ３２５　行データ変形 ＧＢＰＵＴ　$Ｆ３０８　出力ルーチン

解　説　Get Block 2のメインルーチンＧＢＬＫ２（$Ｆ３Ｂ６〜$Ｆ３ＦＢ）と，ＧＢＬＫ２Ｓの関係はＧＣＵＲＳ処理系のＧＣＳＡＶＥとＧＣＳＡＶ１の関係と同じです．Get Block 1では，Ｙ方向のループカウンタとしてＧＢＮＹをそのまま利用しておりましたが，ＧＢＬＫ２ＳではＧＢＮＹを保存しておく必要があるため，代わりのループカウンタとしてＧＢＫを用いております．これは処理が３回繰り返されるためです．同様の理由からこのルーチンの最後でＧＢＩＸのクリアを行っております．

3－6－4 Put Block 1 $1C ($F446)
Put Block 2 $1E ($F589)

Put Block 1では，対角線座標（X 1，Y 1）－（X 2，Y 2）で示される枠内に，指定された色およびファンクションに従って画面のパターンを書き込みます．転送するパターン情報と画面ドット配置の関係は，Get Block 1で示したとおりです．

一方Put Block 2では，枠内に3原色（青・赤・緑）の画面データを書き込みます．

ここで注意しなければならないのは，Put Block 2はGet Block 2に対して完全に逆の機能を果たすのに比較して，Put Block 1はGet Block 1とは必ずしも対になるような機能を持ち合わせてはいないことです．つまり，Get Block 2でメイン側に転送したデータをPut Block 2によって元通りに画面上で復原することは可能ですが，Get Block 1→Put Block 1の場合には必ずしも復原できないわけです．

共有RAM上のパラメータ

アドレス	記　号	内　　容	値
D383 〜 D384	X 1	X 1座標（16ビット）	0～639
D385 D386	Y 1	Y 1座標（16ビット）	0～199
D387 〜 D388	X 2	X 2座標（16ビット）	0～639
D389 〜 D38A	Y 2	Y 2座標（16ビット）	0～199
D38B	CL	カラーコード ※Put Block 2の場合はダミーバイトとなる	0～7
D38C	F	ファンクションコード ※Put Block2の場合，PRESET（コード1）は除く	0～5（SET～NOT）
D38D	Byte	今回転送するバイト数	1～114
D38E D38D＋Byte	Pattern	画面データ	パターン情報

主なサブルーチン

処理系内部

$F4A6	PBINT	初期設定ルーチン
$F50D	PBSHFT	パターンデータ入力／変形
$F56D	PBGET	パラメータ入力／CONTコマンド待ち
$F5B1	PB2SUB	Put Block 2実行ルーチン
$F5D4	PB2LRM	左端および右端の画面データのマスク実行
$F5E1〜$F61F	PB2SET〜PB2EOR	各ファンクションコードに対する画面出力ルーチン（Put Block 2用）
$F62B〜$F658	PB1OR〜PB1EOR	各ファンクションコードに対する画面出力ルーチン（Put Block 1用）

処理系外部

$E121	CONTI	メイン側から継続データ入力（CONTコマンド待ち）
$E3DC	GADDR	グラフィックアドレス生成
$E3F2	GXOFF	除数8の整数除算（Dreg¥8，Dreg mod 8）
$E3FB	LSRD3	Dレジスタを右に3ビットシフト（Dreg¥8）
$E402	BOXINV	対角線座標（X1〜Y2）の読込みと評価
$E46E	FENTRY	ドット表示用フックの初期設定ルーチン

エラー

$E0DB	ERR64	ファンクションコードエラー

補助ワークエリア

$D0A0〜$D0A1	PBNX	枠内のX方向のドット数（X2−X1）［実際にはドット数マイナス1の値］
$D0A2〜$D0A3	PBNY	枠内のY方向のドット数（Y2−Y1）［実際にはドット数マイナス1の値］
$D0A4〜$D0A5	PBVRAM	枠の先頭ドット（X1，Y1）のグラフィックアドレス保存用
$D0A6	PBSP	座標X1に対するビットナンバ（枠の左端）→ PBLM
$D0A7	PBEP	座標X2に対するビットナンバ（枠の右端）→ PBRM
$D0A8	PBLM	枠の左端を表示するとき必要なマスク
$D0A9	PBRM	枠の右端を表示するとき必要なマスク
$D0AA	PBOV	1行あたりの剰余ビットの増分（PBNX mod

		8＋1）
$D0AB 〜$D0AC	PBIB	枠内のX方向のバイト数 （X2¥8－X1¥8）
$D0AD	PBDH	剰余データの先頭ビットのビットナンバ(0〜7)
$D0AE	PBRBEX	PBRBを16ビットデータとみなしたときの上位バイト（常に$00が入る）
$D0AF	PBRB	剰余データのビット数((8－PBDH)mod 8の値)
$D0B0	PBNOT	F＝5(NOT)のとき$FF，それ以外は$00
$D0B1 〜$D0B2	PBBAD	入力バッファ（共有RAM）ポインタ
$D0B3	PBRD	入力バッファ上の残存データのバイト数
$D0B4	PBSAVE	剰余データの保存場所
$D0B5 〜$D0BA	PB1BL，PB1RE，PB1GR	表示用フックの導入により未使用となった領域
$D0BB	PB2ENT	Put Block 2表示用のジャンプベクトル格納場所
$D0BD	PBI	ループカウンタ（Y方向の行数をカウント）
$D0BE 〜$D10E	PBDATA	行データ格納場所（81バイト）

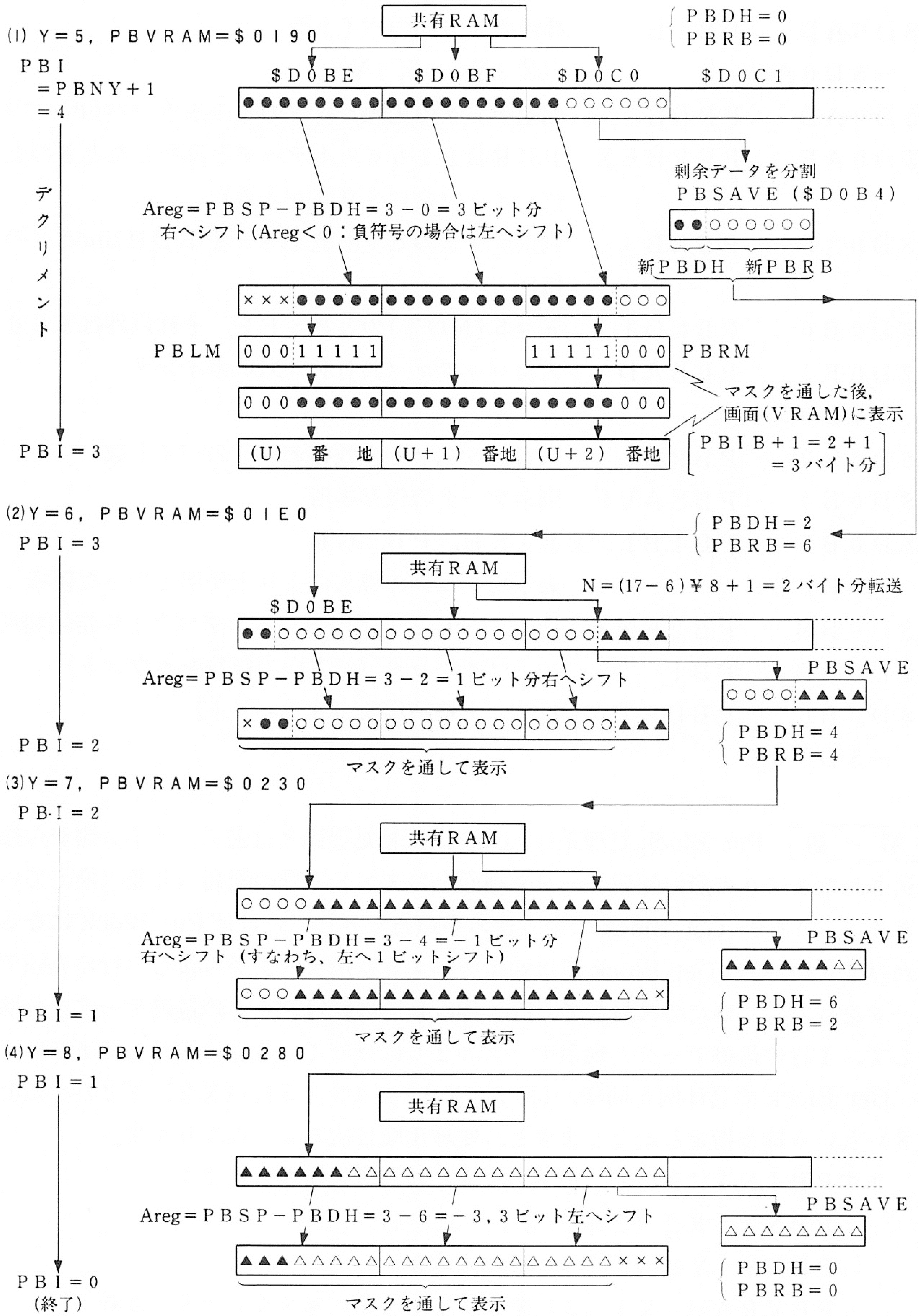
解　説　Put Block 処理系は，Get Block 処理系とは逆に，メイン側から転送された8×nの形の配列データを画面上のx×yの形の配列へと並べ換えていきます．つまり，Get Block の全く逆の手順をたどっていけば Put Block になるわけですが，特に Get Block では剰余データと転送データから新しい行の先頭データを合成していたのに対して，Put Block では転送データの最終データを分解して，1行の最終データと剰余データの2つに分けている点に注目して下さい．

Get Block の具体例と同様，(X1，Y1)＝(3，5)，(X2，Y2)＝(20，8) という枠を指定したとしますと，処理手順は次のようになります．

まずPBINTにおいて，以下のように初期設定がなされます．

PBNX＝X2 (20)－X1 (3)＝17
PBNY＝Y2 (8)－Y1 (5)＝3
PBVRAM＝X1 (3) ¥8＋Y1 (5) ＊$50＝$190
PBSP＝X1 (3) ¥8＝3，PBEP＝X2 (20) ¥8＝4
PBLM＝%00011111，PBRM＝%11111000

図3・6・9　Put Blockにおける行データの入力の入力・変形と表示



ＰＢＯＶ＝（ＰＢＮＸ＋１）￥８＝２

ＰＢＩＢ＝Ｘ２（２０）￥８－Ｘ１（３）￥８＝２－０＝２

ＰＢＤＨ＝ＰＢＲＢＥＸ＝ＰＢＲＢ＝０

そして第１回目の転送で，Ｙ座標＝５に相当する１行分のデータを共有ＲＡＭから＄Ｄ０ＢＥ番地以下に３バイト分転送し（ＰＢＧＥＴを繰り返し呼び出す），変形します．この際最終データをＧＢＳＡＶＥに保存して，次の行に引き継ぎます．ＰＢＬＭ・ＰＢＲＭを通して左端および右端のデータのマスク処理を行った後，１行をＰＢＶＲＡＭ番地以下に出力し，ＰＢＶＲＡＭの値を更新（＄５０を加える）して，ループカウンタＰＢＩをデクリメントし，第２回目の転送（Ｙ＝６)に入ります．なお，ループカウンタは最終の転送の前に，ＰＢＩ＝ＰＢＮＹ＋１（このケースでは，ＰＢＩ＝４）に設定されております．

以下同じ処理をＰＢＩ＝０になるまで続けます．

上記の処理をわかりやすくまとめたのが図３・６・９です．
このうちＰＢＳＨＦＴルーチンが受け持つのは，それぞれの行の転送からマスク処理の手前までです．なおシフトにあたっては，ＧＢＬＫ処理系のような最適化は図られてはおらず，５ビット以上のシフトも行われます．

さて実際の表示を行うにあたっては，カラーコード・ファンクションコードなどが絡んでくるため，この点が少々複雑になります．Put Block １では最初の転送に先立って，Ａレジスタにカラーコード，Ｂレジスタにファンクションコード，Ｘレジスタに Put Block １用ファンクション対応ジャンプテーブルの先頭アドレス（ＰＢ１ＦＮＣ）を入れて表示用初期設定ルーチンＦＥＮＴＲＹを呼び出して，あらかじめＢＬＵＤＳＰ，ＲＥＤＤＳＰ，ＧＲＥＤＳＰの３つのフックに表示ルーチンへのジャンプ命令を書き込んでおきます．ジャンプテーブルＰＢ１ＦＮＣの様子を下表にまとめます．

ジャンプテーブルＰＢＩＦＮＣの様子

アドレス	内　容(ジャンプベクトルの値)		機　能
＄Ｆ４８Ｅ～＄Ｆ４９１	ＰＢ１ＯＲ　(＄Ｆ６２Ｂ)	ＰＢ１ＡＮＤ(＄Ｆ６４１)	ＰＳＥＴ
＄Ｆ４９２～＄Ｆ４９５	ＰＢ１ＯＲ　(＄Ｆ６２Ｂ)	ＰＢ１ＡＮＤ(＄Ｆ６４１)	ＰＲＥＳＥＴ
＄Ｆ４９６～＄Ｆ４９９	ＰＢ１ＯＲ　(＄Ｆ６２Ｂ)	ＰＢ１ＮＯＰ(＄Ｆ６４０)	ＯＲ
＄Ｆ４９Ａ～＄Ｆ４９Ｄ	ＰＢ１ＮＯＰ(＄Ｆ６４０)	ＰＢ１ＡＮＤ(＄Ｆ６４１)	ＡＮＤ
＄Ｆ４９Ｅ～＄Ｆ４Ａ１	ＰＢ１ＥＯＲ(＄Ｆ６５８)	ＰＢ１ＮＯＰ(＄Ｆ６４０)	ＥＯＲ
＄Ｆ４Ａ２～＄Ｆ４Ａ５	ＰＢ１ＯＲ　(＄Ｆ６２Ｂ)	ＰＢ１ＡＮＤ(＄Ｆ６４１)	ＮＯＴ

Put Block ２では，青・赤・緑の各画面ごとに表示を行うので，カラーコード

は関係なく，独自のファンクション別表示ルーチンと，そのベクトルをまとめたジャンプテーブルＰＢ２ＦＮＣ（下表）が用意されております．

ジャンプテーブルＰＢ２ＦＮＣの様子

アドレス	内　容(ジャンプベクトル)	機　　能
＄Ｆ６６Ｅ～＄Ｆ６６Ｆ	ＰＢ２ＳＥＴ(＄Ｆ５Ｅ１)	Ｐ　Ｓ　Ｅ　Ｔ
＄Ｆ６７０～＄Ｆ６７１	ＥＲＲ６４　(＄Ｅ０ＤＢ)	ＰＲＥＳＥＴ
＄Ｆ６７２～＄Ｆ６７３	ＰＢ２ＯＲ　(＄Ｆ６１３)	Ｏ　Ｒ
＄Ｆ６７４～＄Ｆ６７５	ＰＢ２ＡＮＤ(＄Ｆ５ＦＤ)	ＡＮＤ
＄Ｆ６７６～＄Ｆ６７７	ＰＢ２ＥＯＲ(＄Ｆ６１Ｆ)	ＥＯＲ
＄Ｆ６７８～＄Ｆ６７９	ＰＢ２ＳＥＴ(＄Ｆ５Ｅ１)	ＮＯＴ

このベクトルは，ＰＢ２ＥＮＴ上に初期設定されます．

ＰＢＩＮＴ

ＰＢＩＮＴ　　　＄Ｆ４Ａ６　～　＄Ｆ５０Ｃ

機　能

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

入力情報
- Ｘ１　＄Ｄ３８３～４　対角線座標Ｘ１
- Ｙ１　＄Ｄ３８５～６　〃　Ｙ１
- Ｘ２　＄Ｄ３８７～８　〃　Ｘ２
- Ｙ２　＄Ｄ３８９～Ａ　〃　Ｙ２
- Ｆ　＄Ｄ３８Ｂ　ファンクションコード
- Byte　＄Ｄ３８Ｄ　今回転送するバイト数

（以上，共有ＲＡＭ上のパラメータ群）

出力情報　解説参照

ワークエリア
- .ＢＯＸＸ１　＄Ｄ０６９～＄Ｄ０６Ａ
- .ＢＯＸＹ１　＄Ｄ０６Ｂ～＄Ｄ０６Ｃ
- .ＢＯＸＸ２　＄Ｄ０６Ｄ～＄Ｄ０６Ｅ
- .ＢＯＸＹ２　＄Ｄ０６Ｆ～＄Ｄ０７０
- ＢＡＳ　＄Ｅ３９２～＄Ｅ３９９（ビットマスクテーブル）

サブルーチン
- ＢＯＸＩＮＶ　＄Ｅ４０２　対角線座標の読取り・評価
- ＧＡＤＤＲ　＄Ｅ３ＤＣ　グラフィックアドレス生成
- ＧＸＯＦＦ　＄Ｅ３Ｆ２　整数除算（Dreg￥８，Dreg mod ８）
- ＥＲＲ６４　＄Ｅ０ＤＢ　ファンクションコードエラー

解　説　共有RAM上のパラメータ群を読み取って，Put Block 処理系(Put Block 1および2に共通) の補助ワークエリア上に定義されている各種の変数および処理系内定数の初期設定を行います．設定される値は以下のとおりです．

ＰＢＮＸ＝ .ＢＯＸＸ２　－　.ＢＯＸＸ１
ＰＢＮＹ＝ .ＢＯＸＹ２　－　.ＢＯＸＹ１
ＰＢＶＲＡＭ＝座標（ .ＢＯＸＸ１， .ＢＯＹ１）に相当するグラフィックアドレス
ＰＢＳＰ＝ .ＢＯＸＸ１　mod　８
ＰＢＥＰ＝ .ＢＯＸＸ２　mod　８
ＰＢＬＭ＝（ＢＡＳ＋ＰＢＳＰ）の内容×２－１
ＰＢＲＭ＝（ＢＡＳ＋ＰＢＥＰ）の内容×２－１
ＰＢＯＶ＝（ＰＢＮＸ＋１）mod　８
ＰＢＩＢ＝ .ＢＯＸＸ２￥８　－　.ＢＯＸＸ１￥８
ＰＢＤＨ＝ＰＢＲＢＥＸ＝ＰＢＲＢ＝０
ＰＢＮＯＴ…Ｆ＝５（ファンクション＝〝ＮＯＴ〞）のとに＄ＦＦ，
　　　それ以外のコードのとき＄００
ＰＢＲＤ　＝Byteの値（今回転送されたバイト数）
ＰＢＢＡＤ＝＃＄Ｄ３８Ｅ

なおＦ＞５の場合には，ファンクションコードエラー（ＥＲＲ６４）を発生します．

また上記設定エリアのうち，以下にあげるエリアは，処理が終るまでずっと一定の値を保ち続ける処理系内定数（ローカルコンスタント）です．

ＰＢＮＸ，ＰＢＮＹ
ＰＢＮＰ，ＰＢＥＰ
ＰＢＬＭ，ＰＢＲＭ
ＰＢＯＶ
ＰＢＩＢ
ＧＢＲＢＥＸ（ＰＢＲＢの上位８ビットで常に＄００）
ＰＢＮＯＴ

ＰＢＳＨＦＴ

ＰＢＳＨＦＴ　　$ＦＦ５０Ｄ～$ＦＦ５５Ｅ

機能	共有ＲＡＭからの１行分のパラメータの読込み，およびデータのシフト変形を行います．	
レジスタ	Ａ，Ｂ，Ｘ	
入力情報	ＰＢＳＡＶＥ	前の行から引き継いだ剰余データ
	ＰＢＮＸ	枠内Ｘ方向ドット数－１
	ＰＢＲＢＥＸ：ＰＢＲＢ	剰余データのビット数
	ＰＢＳＰ	Ｘ１のビットナンバ
	ＰＢＤＨ	剰余データの先頭ビットのビットナンバ
	ＰＢＩＢ	枠内Ｘ方向バイト数
	ＰＢＯＶ	１行あたりの剰余ビット数の増分
出力情報	ＰＢＤＡＴＡ　$Ｄ０ＢＥ～	１行分の画面出力データ
	ＰＢＳＡＶＥ	次の行に引き継ぐ剰余データ
	ＰＢＤＨ	剰余データの先頭ビットのビットナンバ
	ＰＢＲＢ	剰余データのビット数
サブルーチン	ＬＳＲＤ３　$Ｅ３ＦＢ	Ｄレジスタ３ビット右シフト

解説　ＰＢＧＥＴルーチンを繰り返し呼び出すことによって，１行分のパターンデータをＰＢＤＡＴＡ番地以下に読み込んで，そのデータをＲＯＲ命令もしくはＲＯＬ命令を繰り返し用いて変形していきますが，転送バイト数ならびにシフトドット数の計算式は次に示すとおりです．

転送バイト数　＝｛(ＰＢＮＸ)－（ＰＢＲＢＥＸ：ＰＢＲＢ)｝￥８

シフトドット数＝ＰＢＳＰ－ＰＢＤＨ（符号がプラスならば右シフト，マイナスならば左シフトを表す）

また次の行に引き継ぐ剰余データをＰＢＳＡＶＥに格納し，新しいＰＢＤＨ，ＰＢＲＢを算出します．

ＰＢＤＨ　＝　（ＰＢＤＨ＋ＰＢＯＶ）mod 8

ＰＢＲＢ　＝　（２５６－ＰＢＤＨ）mod 8

すなわち，常にＰＢＤＨ＋ＰＢＲＢ＝８(剰余データがあるとき)，もしくはＰＢＤＨ＋ＰＢＲＢ＝０（剰余データがないとき）のいずれかが成り立っております．

ＰＢＧＥＴ

ＰＢＧＥＴ　　＄Ｆ５６Ｄ　（＄Ｆ５５Ｆ　～　＄Ｆ５８８）

機　能	共有ＲＡＭ入力バッファ上の，ポインタがさす位置から，１バイトのパターンデータを読み取ってＡレジスタに返します．

レジスタ	Ａ，Ｘ（Ｘは保存）	
入力情報	ＰＢＢＡＤ	＄Ｄ０Ｂ１～＄Ｄ０Ｂ２　入力バッファポインタ
	ＰＢＲＤ	＄Ｄ０Ｂ３　入力バッファ上の残存データ数
	ＰＢＮＯＴ	＄Ｄ０Ｂ０　ファンクションＮＯＴフラグ
	..ＣＯＮＴ	＄Ｄ００１　ＣＯＮＴフラグ保存用エリア
出力情報	Ａレジスタ	読込みデータ
	ＰＢＢＡＤ ＰＢＲＤ	Ａレジスタにデータ入力後，更新される．
サブルーチン	ＣＯＮＴＩ	＄Ｅ１２１　メイン側から継続データ入力

解　説　カウンタＰＢＲＤをチェックして，０でなければ，バッファポインタＰＢＢＡＤがさすアドレスから１バイトのデータをＡレジスタに読み込み，カウンタＰＢＲＤをデクリメント，ポインタＰＢＢＡＤをインクリメントします．

ＰＢＲＤ＝０の場合には，ＣＯＮＴフラグ（..ＣＯＮＴの第７ビット）を見て“１”であれば，ＣＯＮＴＩを呼び出してＣＯＮＴコマンド待ちに入り継続データの入力を行いＰＢＢＡＤ＝＄Ｄ３８４，ＰＢＲＤ＝（Byte）に再設定してからＡレジスタへの読込みを行います．一方ＰＢＲＤ＝０で，かつＣＯＮＴフラグが“０”の場合には，Ａレジスタ＝＄００を読込みデータの代わりに返します．

いずれの場合であっても，最後にＰＢＮＯＴフラグをテストして，フラグが立っていたら（ＰＢＮＯＴ≠０），Ａレジスタの値を反転してからリターンします．

ＰＢ２ＳＵＢ

ＰＢ２ＳＵＢ　　＄Ｆ５Ｂ１　～　＄Ｆ５Ｄ３

機　能	Put Block ２において，各色ごとの処理を行うルーチンです．	
レジスタ	Ａ，Ｂ，Ｘ，Ｕ	
入力情報	Ｕレジスタ	ＶＲＡＭアドレス
	ＰＢＮＹ	Ｙ方向のドット数－１
	ＰＢＩＢ	Ｘ方向バイト数

　　PB2ENT　表示用ジャンプベクトル

サブルーチン　PBSHFT

解　説　Put Block 2において，青・赤・緑の各VRAMごとのデータ入力・変形・表示を行うサブルーチンで，GBLK2に対するGBLK2Sと同様の関係が成り立ちます．表示用ベクトルPB2ENTには，PB2SET，PB2OR，PB2AND，PB2EORおよびERR64のいずれかのベクトルが入っており，ファンクションコードで，PRESET（F＝1）を指定した場合には，ベクトルにはERR64（ファンクションコードエラー）が入る一方，NOTを指定した場合には，PB2SETがベクトルとして代入されていますので，表示の際には「JSR　[PB2ENT]」の間接コールにより，いずれかの表示ルーチンを呼び出すことになります．

PB1OR, PB1NOP, PB1AND, PB1EOR

PB1OR	$F62B	～	$F640
PB1NOP	$F640	～	$F640
PB1AND	$F641	～	$F657
PB1EOR	$F658	～	$F66D

機　能　それぞれのファンクションに応じて，実際にVRAMへの出力を行うためのPut Block 1用の表示ルーチン群です．

レジスタ　A，B，X，Y　（Xは保存）

入力情報　Xレジスタ　VRAMアドレス
　　PBIB　出力バイト数
　　PBDATA以下　出力パターンデータ列

解　説　Put Block 1処理系において，実際にVRAMを直接アクセスするサブルーチン群です．処理手順は，PB1NOP（単にRTS命令を行うだけ）を除けば，すべて同等のものであり，マシン語で書くと次のようになります．

```
PB×××   LDB    PBIB+1    バイト数の読出し
         LDY    #PBDATA   出力データの先頭アドレス代入
LNX×××  BITA   VACC      VRAMアクセスフラグON
××LOOP  LDA    B，Y      出力データ読込み
```

```
(COMA           ※ANDの場合のみ)
 ×××A   B, X          ファンクション処理
 STA    B, X          データを出力(表示)
 DECB                 カウンタデクリメント
 BPL    ××LOOP        PBIB+1回のループ
 STA    VACC          アクセスフラグOFF
 RTS
```

上記の×××の部分に, OR, AND, EORのいずれかを入れて読んでみればよくわかるものと思います. なお途中に,「LNX×××」というラベルが混ざっておりますが, これはLINES処理系の出力ルーチンと共通に利用できるように作られているためです(3-5-5節 参照).

なお, いずれの表示ルーチンについても, 両端のマスク処理を行ったあとに呼び出すようになっております.

PB2SET, PB2AND, PB2OR, PB2EOR

PB2SET	$F5E1	~	$F5FC
PB2AND	$F5FD	~	$F612
PB2OR	$F613	~	$F61E
PB2EOR	$F61F	~	$F62A

機　能	Put Block 2用のファンクション別表示ルーチン群	
レジスタ	A, B, X, U　(X, Uは保存)	
入力情報	Bレジスタ	出力データバイト数マイナス1(PBIBの値)
	Xレジスタ	出力データ先頭番地(#PBDATA=$D0BE)
	Uレジスタ	VRAMアドレス[座標(X1, Y)に対応するもの]
	PBLM	左端データ用マスク
	PBRM	右端データ用マスク
	PBDATA以下	出力データ列
サブルーチン	PB2LRM	左端および右端のデータのマスク処理

解　説　PB2SUB用の表示ルーチンです.

PB2ORおよびPB2EORの2つは, PB2LRMルーチンを呼び出して両端のマスク処理を実行した後, PB1OR, PB1EORと同様の手順に従っ

て表示を行います．ただしＶＲＡＭアクセスのＯＮ／ＯＦＦは，上位ルーチンの側で行うようになっている点がＰＢ１系と異るところです．

ＰＢ２ＳＥＴでは，ＰＢ２ＬＲＭ呼び出しの後，マスクＰＢＬＭ・ＰＢＲＭを反転して，ＶＲＡＭの方の両端データのマスク処理をも行った後，ＯＲＡ命令によってＶＲＡＭに書き込んでいきます．

ＰＢ２ＡＮＤでは，ＰＢ２ＬＲＭは用いずにＰＢＬＭと左端データのＯＲ，ＰＢＲＭと右端データのＯＲをとった後，ＡＮＤＡ命令によってＶＲＡＭに書き込んでいきます．ただし，ＰＢ１ＡＮＤとは異りデータの反転は行いません．

ＰＢ２ＬＲＭ

ＰＢ２ＬＲＭ　　＄Ｆ５Ｄ４　～　＄Ｆ５Ｅ０

| 機　能 | Put Block 2の出力データのうち，左端および右端データのマスク処理を実行します． |

| レジスタ | Ａ，Ｂ，Ｘ　（Ｂ，Ｘは保存） |

| 入力情報 |

Ｘレジスタ　　出力データの先頭アドレス（＝＄Ｄ０ＢＥ）
Ｂレジスタ　　出力データの右端の相対位置（＝ＰＢＩＢ）
ＰＢＬＭ　　左端データ用マスク
ＰＢＲＭ　　右端データ用マスク
ＰＢＤＡＴＡ以下　出力データ列

| 解　説 | ＰＢ２ＳＥＴ，ＰＢ２ＯＲ，ＰＢＥＯＲの最初に呼び出されるサブルーチンで，画面に出力しようとしているパターンデータの両端のマスク処理を行います．処理内容は単純で，

（ＰＢＤＡＴＡ）⟵（ＰＢＤＡＴＡ）ＡＮＤ（ＰＢＬＭ）
（Ｂ，Ｘ　）⟵（Ｂ，Ｘ）　ＡＮＤ　（ＰＢＲＭ）

という処理をＡレジスタを使って行います．

3－6－5　Get Character Block 1 $06　($ECF5)
Get Character Block 2 $08　($ECF4)

指定枠内の文字コード (Get Character Block 2の場合には，アトリビュートコードと文字コードを交互に) を読み取って，メイン側に転送します．

共有RAM上のパラメータ

アドレス	記　号	内　　容	値
D383	X1	対角線座標X1	0～79
D384	Y1	対角線座標Y1	0～24
D385	X2	対角線座標X2	0～79
D386	Y2	対角線座標Y2	0～24

※X1≦X2，Y1≦Y2である必要はない (すべてキャラクタ座標)

共有RAM上の復帰情報 (Get Character Block 1)

アドレス	記　号	内　　容	値
D383	Byte	今回転送されるバイト数	1～124
D384 ～ D383+Byte	C_m	転送文字列 (1≦m≦124)	文字コードの列 C_1，C_2……C_m

共有RAM上の復帰情報 (Get Character Block 2)

アドレス	記　号	内　　容	値
D383	Byte	今回転送されるバイト数	1～124
D384 ～ D383+Byte	A_m，C_m	転送コードの列 (1≦m≦62)	A_1，C_1，A_2，C_2，…………，A_m，C_m

主なサブルーチン

処理系内部

$ED0F　CGBDT　共有RAM出力バッファに1バイト出力

$ED23　CBINIT　初期設定ルーチン(Put Character Block も共通)

$ED54　CBLADR　ポインタ更新，終了判定 (同　上)

処理系外部

$E0F5　CONTO　メイン側に継続データ出力

$E20C　CAXCR　キャラクタ座標のチェック(3－4－7節参照)

$E27D　ASSFLD　各画面の先頭にフィールドのはじまりを設定

＄E3D5　BADR　バッファアドレス生成

エラー

＄E0D2　ERR61　コンソール座標（キャラクタ座標）エラー

補助ワークエリア

＄D0A0　CBCMD　コマンドの判別用

＄D0A1　CBY1　対角線座標Y1（座標ポインタYを兼ねる）

＄D0A2　CBX1　対角線座標X1（座標ポインタXを兼ねる）

＄D0A3　CBY2　対角線座標Y2

＄D0A4　CBX2　対角線座標X2

※CBY1，CBX1には，Min（Y1，Y2），Min（X1，X2）の値が，CBY2，CBX2には，Max（Y1，Y2），Min（X1，X2）の値が，それぞれ格納されます．

＄D0A5　CBXTOP　対角線座標X1の保存用エリア

解　説　Get Character Block（以後CGBLKと略記）処理系は，CGBLK1およびCGBLK2の2つのコマンド処理を行う共用ルーチンになっており，どちらのコマンドであるかは，処理の最初にCBCMDに刻印されます（CGBLK1：＄00，CGBLK2：＄4F）．CBINITルーチンを呼び出してCBY1～CBXTOPまでの初期設定を行い，Uレジスタに出力バッファ（共有RAM上）の先頭アドレス（＝＄D384）を入れ，Byte（＄D383）をクリアした後メインループに入ります．

メインループではCBLADRを呼び出して，現在の座標（CBX1，CBY1）に対するバッファアドレスを求め，コンソールバッファから1文字分のデータを読み取ります．この際CBCMDをチェックして，＄4F（CGBLK2）であれば，アトリビュートコード・文字コードの順に読み出してそれぞれCGBDTで出力するのに対し，＄00であれば，アトリビュートコードの読出しと出力をスキップして文字コードのみ読み出して出力します．この後ループの先頭に戻って同じ処理を繰り返します．なお，CBLADRルーチンでは，座標の更新・終了判定なども同時に行うようになっており，終了（CBY1＞CBY2）の場合にはCBLENDに入って，リターンアドレスをつぶし，ASSFLDにジャンプします．

Put Character Block 1　$ 0 7　($ E D 7 2)
Put Character Block 2　$ 0 9　($ E D 7 1)

Get Character Block 処理系とは逆の機能を持つコマンドであり，Get――処理系と同様，２つのコマンドの共用ルーチンになっています．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
D 3 8 3	X 1	対角線座標X 1	0 ～ 7 9
D 3 8 4	Y 1	対角線座標Y 1	0 ～ 2 4
D 3 8 5	X 2	対角線座標X 2	0 ～ 7 9
D 3 8 6	Y 2	対角線座標Y 2	0 ～ 2 4
D 3 8 7	A T	アトリビュート(色指定)	C P B L K 2 は不要
D 3 8 8	Byte	今回転送するバイト数	
D 3 8 9	$(A_m,)C_m$	転送コードの列	$(A_1,)\ C_1,\ \cdots,\ (A_m,)\ C_m$
D388＋Byte		※C P B L K 1 の場合は，C_m だけとなる	

主なサブルーチン

処理系内部

$ED23　CBINIT　初期設定
$ED54　CBLADR　バッファアドレス生成など
$EDA5　CPBDT　共有RAM入力バッファからの1バイト読込み

処理系外部

$E121　CONTI　メイン側から継続データ入力
$E20C　BADR　バッファアドレス生成
$E225　BVAD　VRAMアドレス生成
$E27D　ASSFLD　フィールドマークの設定など
$E3D5　CAXCR　キャラクタ座標のチェック
$E51D　CDSP　キャラクタ1文字表示

エラー

$E0D2　ERR61　コンソール座標エラー

補助ワークエリア

Get Character Block 処理系のものに加え，さらに以下のものを使用．

＄Ｄ０Ａ６　　ＣＰＢＡＴ　　アトリビュート保存用
＄Ｄ０Ａ７　　ＣＰＢＣＮＴ　入力バッファのカウンタ
＄Ｄ０Ａ８　　ＣＢＳＡＶＥ　現在の座標（Ｘ，Ｙ）の保存用
　−＄Ｄ０Ａ９

解　説　Get Character Blockとは全く逆の機能を持っている処理系であり，やはり，２つのコマンドの共用ルーチンとなっています．Put Character Block 1（ＣＰＢＬＫ１）では，ＡＴによって一括してアトリビュートを指定してしまうのに対して，Put Character Block 2（ＣＰＢＬＫ２）では，一つひとつの文字についてアトリビュートを指定していく点が主な違いです．

ＣＢＩＮＩＴ

ＣＢＩＮＩＴ　　＄ＥＤ２３　〜　＄ＥＤ５３

機　能　Get Character BlockおよびPut Character Block処理用の初期設定を行います．

レジスタ　Ａ，Ｂ

入力情報　共有ＲＡＭ上のパラメータＸ１〜Ｙ２（＄Ｄ３８３〜＄Ｄ３８７）

出力情報
ＣＢＹ１　＄Ｄ０Ａ１　Min ｛Ｙ１，Ｙ２｝…対角線座標右上
ＣＢＸ１　＄Ｄ０Ａ２　Min ｛Ｘ１，Ｘ２｝…対角線座標右上
ＣＢＹ２　＄Ｄ０Ａ３　Max ｛Ｙ１，Ｙ２｝…対角線座標左下
ＣＢＸ２　＄Ｄ０Ａ４　Max ｛Ｘ１，Ｘ２｝…対象線座標右下
ＣＢＸＴＯＰ　＄Ｄ０Ａ５　ＣＢＸ１の値がコピーされる

サブルーチン
ＣＡＸＣＲ　＄Ｅ３Ｄ５　キャラクタ座標のチェック（誤りでないときＣフラグセット）
ＥＲＲ６１　＄Ｅ０Ｄ２　コンソール座標エラー

解　説　このルーチンで設定される値は上記のとおりです．共有ＲＡＭから読み取った座標が誤りであればＥＲＲ６１に分岐します．

ＣＢＬＡＤＲ

ＣＢＬＡＤＲ　　＄Ｅ０５４　〜　＄Ｅ０７０

機　能　座標（ＧＢＸ１，ＧＢＹ１）のセーブ，バッファアドレスのロード，

ＧＢＸ１，ＧＢＹ１の更新，終了判定などを実行します．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ		
入力情報	ＣＢＹ１	＄Ｄ０Ａ１	現在の座標
	ＣＢＸ１	＄Ｄ０Ａ２	
出力情報	ＣＢＹ１	＄Ｄ０Ａ１	更新された座標
	ＣＢＸ１	＄Ｄ０Ａ２	
	ＣＢＳＡＶＥ	＄Ｄ０Ａ８～＄Ｄ０Ａ９	現在の座標のセーブ値
	Ｘレジスタ	文字コードバッファアドレス	
	Ｙレジスタ	アトリビュートバッファアドレス	
ワークエリア	ＣＢＸＴＯＰ	＄Ｄ０Ａ５	Ｘ１の保存値
サブルーチン	ＢＡＤＲ	＄Ｅ２０Ｃ	バッファアドレスロード［Ｘレジスタ：文字コードバッファアドレス　Ｙレジスタ：アトリビュートバッファアドレス］
	ＣＢＬＥＮＤ	＄ＥＤ６Ｃ	コマンド終了（リターンアドレスを１つぶしてＡＳＳＦＬＤに分岐）
	ＡＳＳＦＬＤ	＄Ｅ２７Ｄ	各画面の先頭にフィールドのはじまりを設定

解　　説　このルーチンでは，まず現在の座標をＣＢＳＡＶＥにセーブした後，ＣＢＹ１（現在のＹ座標）とＣＢＹ２とを比較します．もしもＣＢＹ１＞ＣＢＹ２ならば，ＣＢＬＥＮＤに分岐してコマンド処理を終了させます．もしＣＢＹ１≦ＣＢＹ２ならば，ＢＡＤＲをコールしてＸおよびＹレジスタにバッファアドレスを求めたのち，座標の値を更新します．つまりＣＢＸ１＜ＣＢＸ２だった場合には，ＣＢＸ１をインクリメントし，ＣＢＸ１＝ＣＢＸ２だった場合には，次の行へ進むようＣＢＸ１＝ＣＢＸＴＯＰとし，ＣＢＹ１をインクリメントします．

ＣＧＢＤＴ

ＣＧＢＤＴ　　＄ＥＤ０Ｆ ～ ＄ＥＤ２２

機　　能　Get Character Block 処理系のサブルーチンで，共有ＲＡＭへＡレジスタの値を格納（出力）します．共有ＲＡＭが満杯の場合には，ＣＯＮＴコマンドの受付け（ＣＯＮＴＯ）を行い，メイン側に共有ＲＡＭ上のデータを読み取らせた後でデータの格納を行います．

レジスタ　Ａ，Ｕ

入力情報　Aレジスタ　　共有RAMへの出力データ

ワークエリア　共有RAM上のパラメータ Byte（$D383）

Uレジスタ　　共有RAM用出力バッファポインタ

サブルーチン　CONTO　　$E0F5　メイン側への継続データ出力

解　説　まずポインタUをチェックして，共有RAMが満杯になっていない場合には，Aレジスタの値を（U）番地に格納してポインタUをインクリメントすると同時に，共有RAM上のバイトカウンタ Byte のインクリメントも行います．

共有RAMが満杯だった場合には，CONTOをコールし，Uレジスタに共有RAM先頭データのアドレス（$D384）の値を入れ，カウンタ Byte をクリアしてからAレジスタの格納を行います．

CPBDT

CPBDT　　　$ED7E　～　$EDBB

機　能　Put Character Block 処理のサブルーチンで，共有RAMからAレジスタへの値の読出し（入力）を行います．

レジスタ　A，U

入力情報　..CONT　　$D001　CONTフラグ保存用エリア

ワークエリア　Uレジスタ　　共有RAM入力バッファポインタ

CPBCNT　共有RAM上に残存するデータのバイト数

サブルーチン　CONTI　　$E121　メイン側からの継続データ入力

解　説　このルーチンではCPBCNTをデクリメントして，共有RAM上にまだデータが残存する場合（デクリメント後のCPBCNT≧0のとき）には，バッファポインタUが示す番地からAレジスタへの値の読込みを行います．もしもCPBCNT＜0の場合には，CONTIを呼び出してCONTコマンド待ちを行い，Uレジスタにデータバッファ先頭アドレス（$D384）を，またCPBCNTには，共有RAMパラメータ Byte（$D383）の値をそれぞれ設定した後，データの読込みを実行します．ただしCONTIを呼び出す前にCONTフラグ保存用エリア..CONT（$D001）のビット7をチェックして，メイン側からの継続データがあるかどうかをあらかじめ確認し，継続データがない場合（ビット7＝0）にはCBLENDに分岐して，即座にコマンド処理を終了させます．

3－6－6　Change　Color　＄１A　（＄F２１B）

対角線座標で示される枠内の色を変更（枠内にある旧カラーコードのドットをすべて指定した新カラーコードのドットに変更）します．変更する色の数は最大8色まで指定できます．

共有RAM上のパラメータ

アドレス	記　号	内　　　容	値
D383 D384	X1	対角線座標X1	0～639 (グラフィック座標)
D385 D386	Y1	対角線座標Y1	0～199 (グラフィック座標)
D387 D388	X2	対角線座標X2	0～639 (グラフィック座標)
D389 D38A	Y2	対角線座標Y2	0～199 (グラフィック座標)
D38B	N	変更するカラーの数	1～8
D38C	CL_1	旧カラーコード1	0～7
D38D	CL_1	新カラーコード1	0～7
⋮	⋮	⋮	⋮
D38A＋2n	CL_n	旧カラーコードn	0～7
D38B＋2n	CL_n	新カラーコードn	0～7

主なサブルーチン

＄E402	BOXINV	対角線座標の読取り・評価・並べ換え
＄E3DC	GADDR	グラフィックアドレスの生成
＄E39A	GPGETR	指定位置のドットのカラーコード読取り
＄E469	GPDSPI	グラフィックドット表示用初期化ルーチン
＄E497	GPDSPR	グラフィックドット表示ルーチン
＄F280	CHANGX	ERR67（指定色数エラー）発生

補助ワークエリア

＄Ｄ０Ａ０ ～＄Ｄ０Ａ７	ＣＨＧＣＬ	カラー対応テーブル（旧カラーコードと新カラーコードの対応づけを記憶するテーブル；下図参照）
＄Ｄ０Ａ８	ＣＨＧＩ	変更するカラー数をかぞえるループカウンタ（カラー対応テーブル作成用）
＄Ｄ０Ａ９	ＣＨＧＪ	枠のＹ方向のループカウンタ（Ｙ方向の進行状況をカウントする）
＄Ｄ０ＡＡ	ＣＨＧＫ	枠のＹ方向の大きさ（Ｙ２－Ｙ１）を記憶しておくエリア

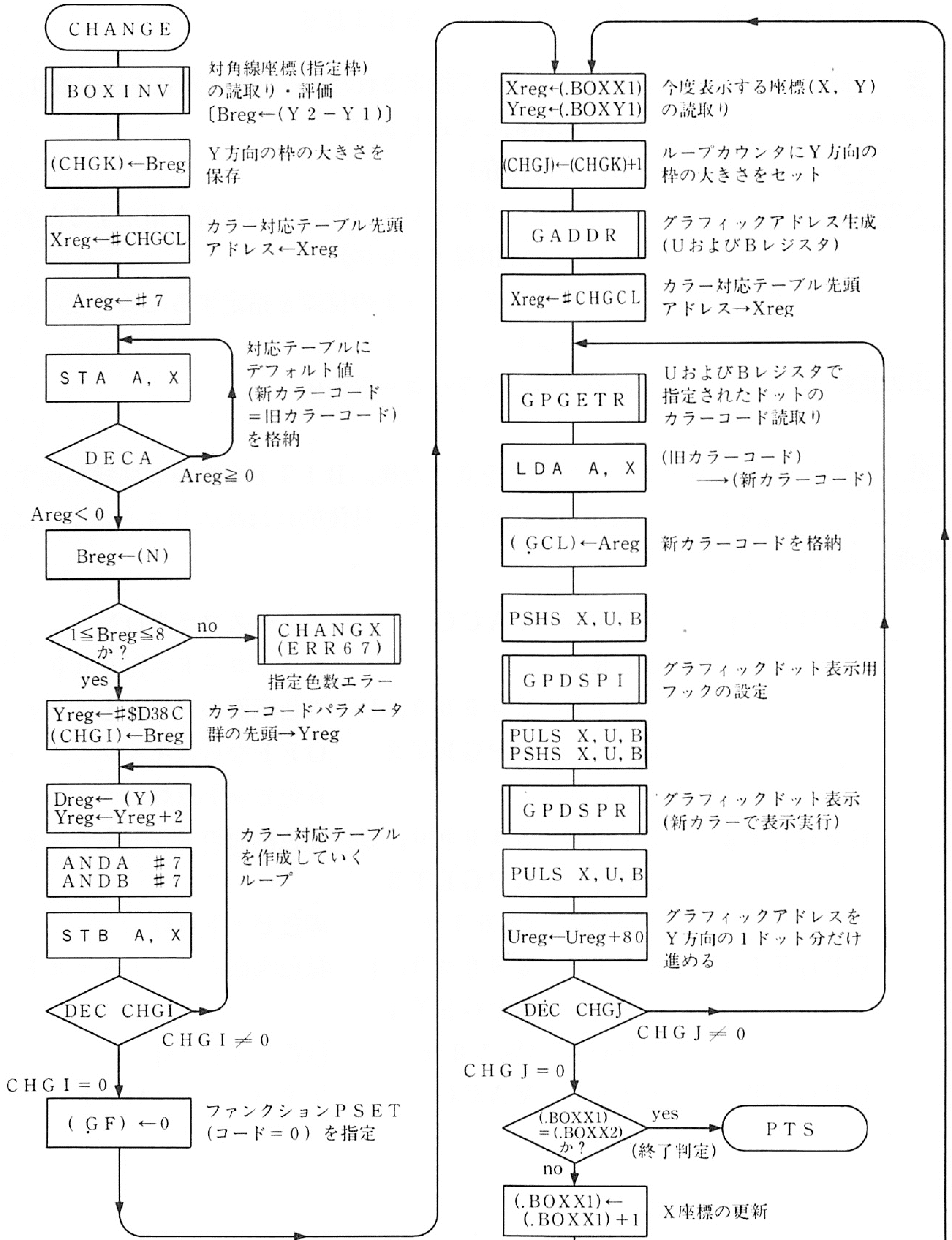
解　説　この処理系（ＣＨＡＮＧＥ）の鍵は，カラー対応テーブルの存在です．まずＢＯＸＩＮＶルーチンによって対角線座標の読取りを行った後，次のようなカラー対応テーブルをＣＨＧＣＬ（＄Ｄ０Ａ０）以下に作成します．

アドレス	旧カラーコード	値(対応する新カラーコードが入る)	
Ｄ０Ａ０	0 (黒色)相当	ＣＬｉ	共有ＲＡＭ上のパラメータによって指定されたものについては，その指定色コードが，指定されなかったものについては，旧カラーコードに等しい値がそれぞれ格納される．
Ｄ０Ａ１	1 (青色)相当	ＣＬｊ	
Ｄ０Ａ２	2 (赤色)相当	ＣＬｋ	
Ｄ０Ａ３	3 (紫色)相当	⋮	
Ｄ０Ａ４	4 (緑色)相当	⋮	
Ｄ０Ａ５	5 (水色)相当	⋮	
Ｄ０Ａ６	6 (黄色)相当	⋮	
Ｄ０Ａ７	7 (白色)相当	ＣＬｍ	

この後，指定枠内の一つひとつのドットごとにＧＰＧＥＴＲルーチンを用いて各ドットのカラーコード（その時点で画面に表示されている色のコード）を読み取り，上記のテーブルを参照して旧カラーコードに対応する新カラーコードをグラフィック表示色レジスタ．ＧＣＬに格納します．そしてＧＰＤＳＰＩおよびＧＰＤＳＰＲルーチンを続けて呼び出して，一つのドットに関する表示色変更を終えます．この一連の処理をポインタ類を更新しながら繰り返し，枠内のすべての表示色変更を遂行します．

それではChange Color処理系のフローチャートを図３・６・10に掲げます．

図３・６・10　Change Color 処理系のフローチャート



ＧＰＧＥＴＲ

ＧＰＧＥＴＲ　　$E39A ～ $E3B6

機　能　UおよびBレジスタによって指定されたドットの表示色を読み取り，そのカラーコードをAレジスタに格納して返します．

レジスタ　A，B，U（B，Uは保存）

入力情報　Uレジスタ　グラフィックアドレス（ドットの位置を指定するためのVRAM相対アドレス）

Bレジスタ　ビットマスク（ドットの位置を指定するためのビットパターン）

出力情報　Aレジスタ　読み取ったカラーコードの値

解　説　VRAMアクセスフラグを立てた後，BITB命令を3回繰り返すことによって画面ドットの表示色を識別します．具体的には次のリストのように処理しております．

```
GPGETR  BITA  VACC      アクセスフラグON
        CLRA            カラーコード＝%000
        BITB  $0000,U   青色画面のドットをテスト
        BEQ   GPGET2    OFFならスキップ
        INCA            青色ビット点灯
GPGET2  BITB  $4000,U   赤色画面のドットをテスト
        BEQ   GPGET3
        ADDA  #%010     赤色ビット点灯
GPGET3  BITB  $8000,U   緑色画面のドットをテスト
        BEQ   GPGET4
        ADDA  #%100     緑色ビット点灯
GPGET4  STA   VACC      アクセスフラグOFF
        RTS
```

３－６－７　ブロックデータ処理汎用サブルーチン

ＢＯＸＩＮＶ

ＢＯＸＩＮＶ　　$E402 ～ $E419

| 機　能 | 共有ＲＡＭ上のパラメータＸ１～Ｙ２（＄Ｄ３８３～＄Ｄ３８Ａ番地のグラフィックアドレス群)を読み取って，ワークエリア．ＢＯＸＸ１～．ＢＯＹＹ２（＄Ｄ０６９～＄Ｄ０７０）に設定するとともに，枠の大きさをレジスタに入れて返します． |

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ
入力情報	共有ＲＡＭ上のパラメータＸ１～Ｙ２
出力情報	．ＢＯＸＸ１　＄Ｄ０６９～＄Ｄ０６Ａ　Min｛Ｘ１，Ｘ２｝の値
	．ＢＯＸＹ１　＄Ｄ０６Ｂ～＄Ｄ０６Ｃ　Min｛Ｙ１，Ｙ２｝の値
	．ＢＯＸＸ２　＄Ｄ０６Ｄ～＄Ｄ０６Ｅ　Max｛Ｘ１，Ｘ２｝の値
	．ＢＯＸＹ２　＄Ｄ０６Ｆ～＄Ｄ０７０　Max｛Ｙ１，Ｙ２｝の値
	Ｘレジスタ　枠のＸ方向の大きさ（．ＢＯＸＸ２－．ＢＯＸＸ１）
	Ｄレジスタ　枠のＹ方向の大きさ（．ＢＯＸＹ２－．ＢＯＸＹ１）
サブルーチン	ＭＯＶＥ　＄Ｅ４４Ａ　ブロック転送ルーチン
	ＢＯＸ　＄Ｅ４１Ａ　対角線座標のチェック・並べ換え

解　説　対角線座標（Ｘ１，Ｙ１)－（Ｘ２，Ｙ２）によってグラフィック画面の枠を指定するコマンド（ただしＬＩＮＥコマンドは除く）では，共有ＲＡＭ上の＄Ｄ３８３～＄Ｄ３８Ａ番地の８バイトに座標指定用のパラメータを置くようになっております．このルーチンでは，まずこれらのパラメータをＭＯＶＥサブルーチンによって，ワークエリアの．ＢＯＸＸ１～．ＢＯＸＹ２に転送します．ただこのままの形だと，メイン側からの指定によって大小関係の順序がバラバラである可能性があり（Ｘ１≦Ｘ２かつＹ１≦Ｙ２，Ｘ１＞Ｘ２かつＹ１≦Ｙ２，………などの４種類が考えられる)，後の処理がやりづらいので，並べ換えを行うことにより順序をそろえます．並べ換えを実行するのはＢＯＸサブルーチンです．最後に，Ｘレジスタに．ＢＯＸＸ２－．ＢＯＸＸ１の値を，またＤレジスタに．ＢＯＸＹ２－．ＢＯＸＹ１の値を，それぞれ代入して，上位ルーチンにリターンします．

ＢＯＸ

ＢＯＸ　　　　　＄Ｅ４１Ａ　～　＄Ｅ４４１
ＢＯＸＲ　　　　＄Ｅ４１Ｄ

機　能　対角線座標のチェックおよび並べ換えを行います．

レジスタ　Ａ，Ｂ，Ｘ，Ｙ，Ｕ

入力情報　．ＢＯＸＸ１～．ＢＯＸＹ２［ＢＯＸエントリの場合］
Ｕレジスタおよび（Ｕ）～（Ｕ＋７）番地［ＢＯＸＲエントリ］

出力情報　上記の入力情報の座標データを，順序をそろえて並べ換えたもの

サブルーチン　ＧＡＸＣＨＫ　＄Ｅ４４２　グラフィック座標のチェック
（誤りの場合にはＥＲＲ６３発生）

解　説　ＢＯＸＩＮＶのところで説明したとおり，Ｘ１～Ｙ２の座標データが不揃いの場合には並べ換えを行って順序を揃えます．並べ換えの結果，座標データは次のようになります．

．ＢＯＸＸ１＝Min｛．ＢＯＸＸ１，．ＢＯＸＸ２｝
．ＢＯＸＹ１＝Min｛．ＢＯＸＹ１，．ＢＯＸＹ２｝
．ＢＯＸＸ２＝Max｛．ＢＯＸＸ１，．ＢＯＸＸ２｝
．ＢＯＸＹ２＝Max｛．ＢＯＸＹ１，．ＢＯＸＹ２｝

※　Min｛ｘ，ｙ｝は，ｘ，ｙのうち大きくない方を値とする関数
Max｛ｘ，ｙ｝は，ｘ，ｙのうち小さくない方を値とする関数

ただし，ＢＯＸＲエントリの場合にはＸ１～Ｙ２の座標データが，Ｕレジスタで指示される番地以降に格納されているものとして処理を進めます．

ＧＡＸＣＨＫ

ＧＡＸＣＨＫ　　　＄Ｅ４４２　～　＄Ｅ４４９

機　能　ＧＡＸＣＲルーチンを呼び出して，グラフィック座標（Ｘ，Ｙ）に誤りがあった場合には，ＥＲＲ６３（＄Ｅ０Ｄ８，グラフィック座標エラー）を発生します.

レジスタ　Ｘ，Ｙ

入力情報　Ｘレジスタ　グラフィックＸ座標
　　　　　　Ｘレジスタ　グラフィックＹ座標

ＧＡＸＣ

ＧＡＸＣ　　　　＄Ｅ３８３　～　＄Ｅ３９１
ＧＡＸＣＲ　　　＄Ｅ３８８

機　能　グラフィック座標（Ｘ，Ｙ）のチェックを行って，その結果をキャリーフラグ（Ｃフラグ）に格納します.

レジスタ　Ｘ，Ｙ

入力情報　.ＧＸおよび .ＧＹにそれぞれ座標Ｘ，Ｙ［ＧＡＸＣエントリ］
　　　　　　Ｘ，Ｙの各レジスタにそれぞれ座標Ｘ，Ｙ［ＧＡＸＣＲエントリ］

出力情報　Ｃフラグ　　Ｘ＜６４０かつＹ＜２００の場合にはセット
　　　　　　　　　　　　Ｘ≧６４０またはＹ≧２００の場合にはクリア

ＭＯＶＥ

ＭＯＶＥ　　　　＄Ｅ４４Ａ　～　＄Ｅ４５１

機　能　ブロック転送を行います（ ,Ｘ＋　──→　 ,Ｙ＋　の方向の転送).

レジスタ　Ｂ，Ｘ，Ｙ

入力情報　Ｘレジスタ　転送元（Source　　）先頭アドレス
　　　　　　Ｙレジスタ　転送先（Destination）先頭アドレス
　　　　　　Ｂレジスタ　転送バイト数（０を代入して呼び出すと，２５６バイト分の転送が行われるので注意を要する）

※　Ｘ＜Ｙ＜Ｘ＋Ｂの場合には，データの消失が起こるので要注意!!

３－７　キーボード・タイマとその割込み処理

３－７－１　ＩＮＫＥＹ　　＄２９　（＄ＦＤ５Ｃ）

キーボード（キー入力バッファ）より，キーコードを読み取ります．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
Ｄ３８３	ＣＦ	制御フラグ	Bit ０：ウェイトフラグ＝１のときウェイト Bit １：ＲＥＳＥＴ＝１のときバッファをクリア

共有ＲＡＭ上の復帰情報

アドレス	記　号	内　　容	値
Ｄ３８３	Ｋ	入力キーコード	０～２５５
Ｄ３８４	ＥＮ	入力の有無	０：なし　１：あり

主なサブルーチン

＄ＦＤ７０	ＫＥＹＩＮ	キー入力バッファからキーコードを読み取る
＄ＦＤ９Ｅ	ＫＢＦＥＲＡ	キー入力バッファをクリアする
＄ＦＤＡＣ	ＫＥＹＩＲＱ	キー割込みを処理する
＄ＦＤＥ６	ＫＳＴＢＵＦ	キー入力バッファへキーコードを格納する

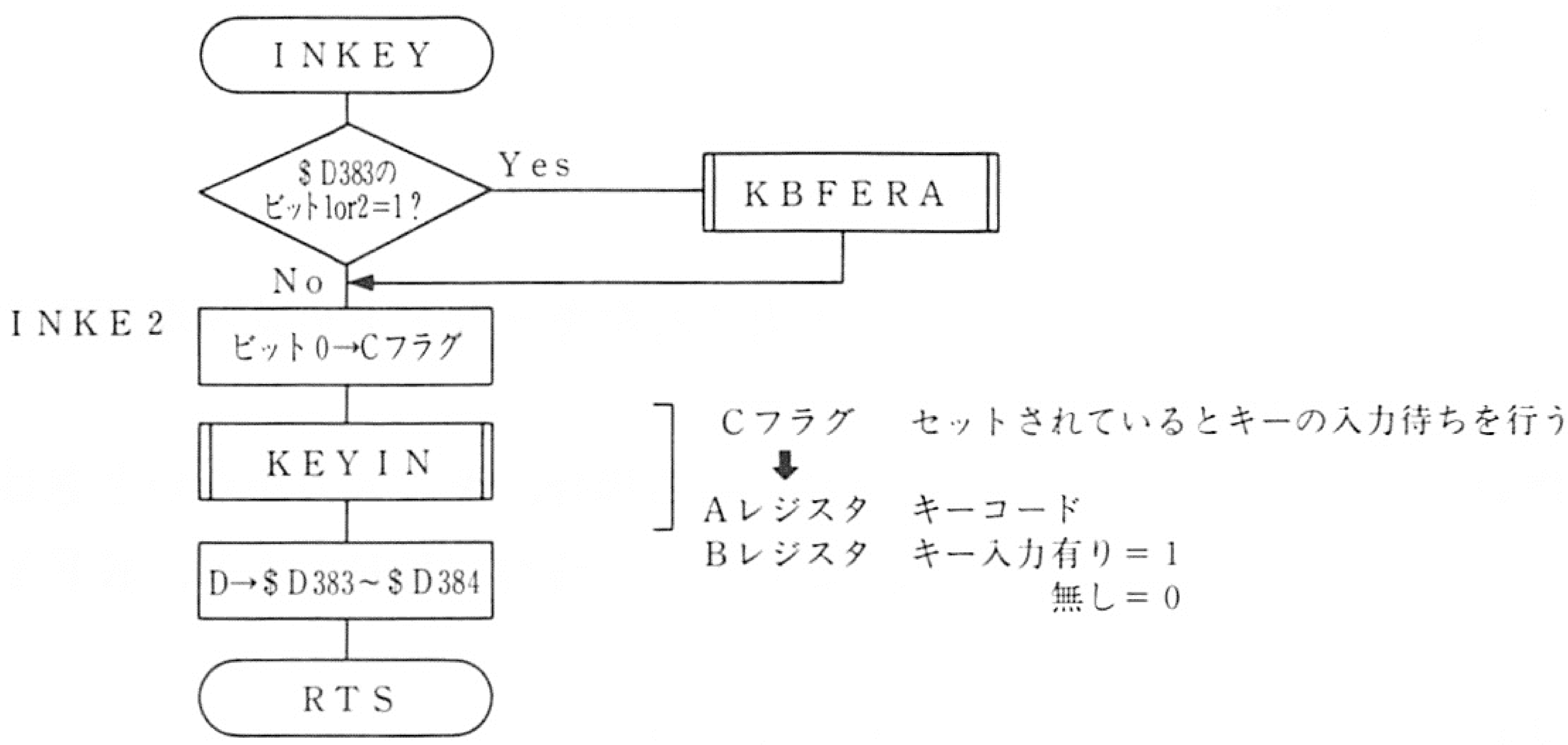
解　説　動作内容は次のようになります．フローチャートを図３・７・１にあげました．

１）ＣＦのビット１もしくはビット２のどちらかが１のときは，ＫＢＦＥＲＡを使用して，キー入力バッファをクリアします．

２）ＣＦのビット０の内容をＣフラグに代入して，ＫＥＹＩＮを呼び出します（ＫＥＹＩＮではＣフラグを参照して，キー入力待ちの判断をします）．

３）ＫＥＹＩＮから復帰情報として，Ａレジスタに入力キーコード，Ｂレジスタに入力の有無が戻ってきますので，それを＄Ｄ３８３～＄Ｄ３８４のＫとＥＮに代入して終了します．

図3・7・1　ＩＮＫＥＹの処理



ＫＥＹの定義

ＰＦキーは含みます．ＳＨＩＦＴ，ＣＡＰＳ，カナ，ＣＴＲＬのキーは除きます．また，Break キーも除きます．

ＩＮＫＥＹ処理系で使用するＩ／Ｏポートとワークエリア

$Ｄ４００	ＫＣＯＤ１	ＭＳＢ（ビット７）＝１　ＰＦキー ＝０　普通のキー
$Ｄ４０１	ＫＣＯＤ２	ＰＦキーのときは，そのＰＦキーの番号 普通のキーのときはキーコード
$Ｄ００２	ＫＬＯＣＫ	キー入力禁止フラグ＝０キー入力受け付ける ≠０キー入力を受け付けない
$Ｄ００３	ＫＢＦＦＬＧ	キーバッファフラグ＝０ノンバッファモード ≠０バッファモード
$Ｄ００４	ＫＣＯＵＮＴ	キー入力数　読み取れていない入力キーコードの数（ノンバッファモードのときは，０か１）
$Ｄ００５	ＫＨＥＡＤ	キー入力バッファ読取りポインタ　次にキーコードを読み込むべきアドレス（キー入力バッファ内）
$Ｄ００７	ＫＴＡＩＬ	キー入力バッファ書込み用ポインタ　次にキーコードを書き込むべきアドレス（キー入力バッファ内）

キーが入力されると，キーボードインタフェースＬＳＩがＦＩＲＱ割込みを発生させるため，ＦＩＲＱの処理としてキー入力処理ルーチンがキーバッファにキーコードを蓄えていく構造になっています．

ＫＥＹＩＮ

ＫＥＹＧＥＴ	＄ＦＤ６Ｅ～＄ＦＤ９４
ＫＥＹＩＮ	＄ＦＤ７０
ＫＥＹＩＮ０	＄ＦＤ７６

機　　能	キー入力バッファから，入力されたキーコードを読み取ります．		
レジスタ	Ａ，Ｂ，Ｘ		
入力情報	Ｃフラグ	セットされている場合，キー入力がないときは，キーが入力されるまで待つ（ＫＥＹＩＮ，ＫＥＹＩＮ０）	
出力情報	Ａレジスタ	入力されたキーコード	
	Ｂレジスタ	キー入力がある場合は，＄０１が代入されます キー入力がない場合は，＄００が代入されます	
ワークエリア	..ＡＢＲＴ	＄Ｄ０００	アボートフラグ
	ＫＬＯＣＫ	＄Ｄ００２	キー入力禁止フラグ
	ＫＣＯＵＮＴ	＄Ｄ００４	キー入力数
	ＫＨＥＡＤ	＄Ｄ００５～＄Ｄ００６	キー入力バッファ読取り用ポインタ
サブルーチン	ＩＮＣＫＢＸ	＄ＦＤ９５	キー入力用のポインタ更新

解　　説　エントリの違いによって，次のような相違があります．

1）ＫＥＹＧＥＴ　必ずキー入力待ちを行います．キー入力のときに，キーを必ずＵＮＬＯＣＫ状態（キー入力を受け付ける）にします．

2）ＫＥＹＩＮ　Ｃフラグがセットされていればキー入力待ちを行います．キー入力時には，必ずＵＮＬＯＣＫ状態にします．

3）ＫＥＹＩＮ０　Ｃフラグがセットされていればキー入力待ちを行います．

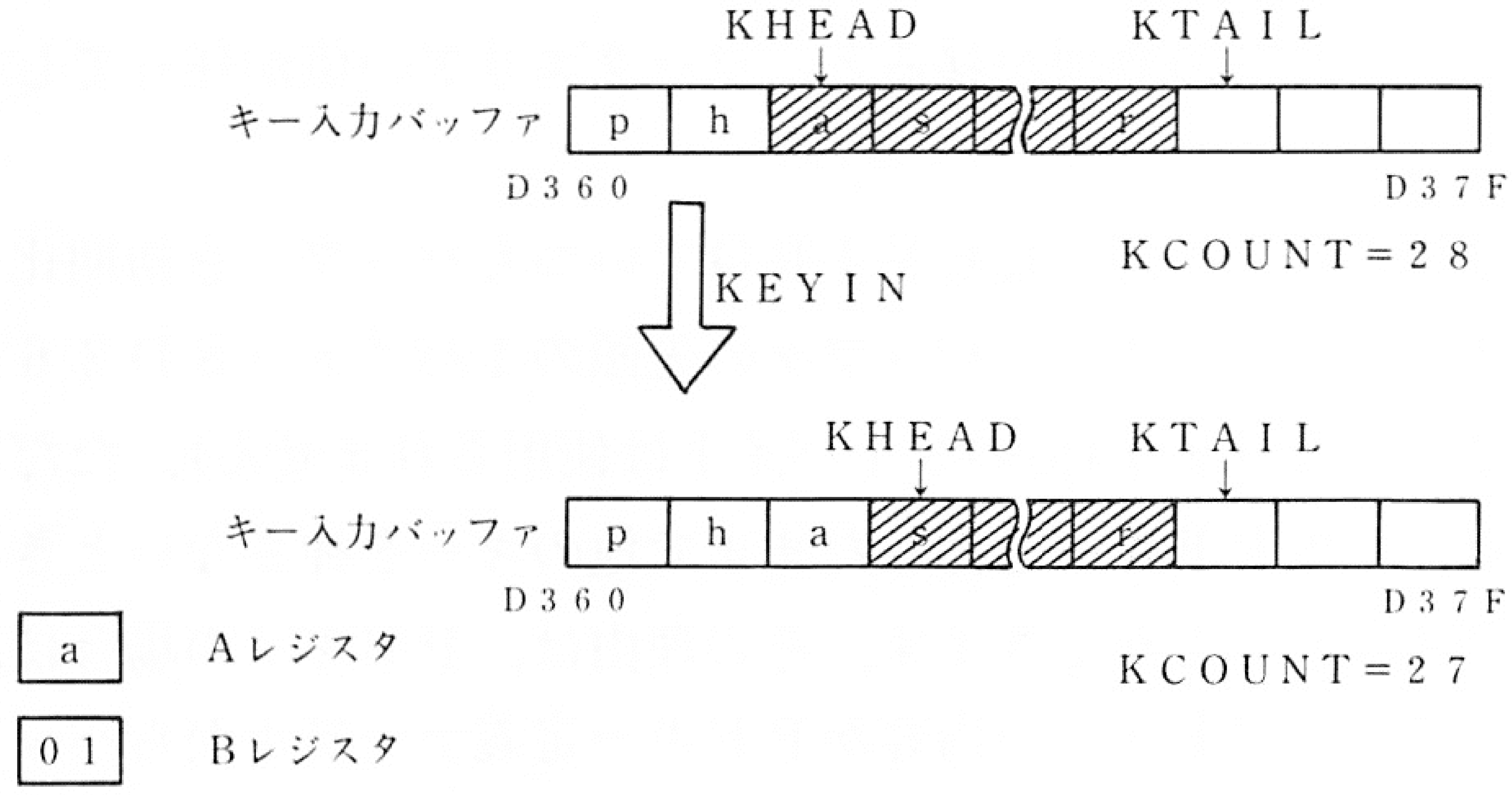
動作の概要は，次の様になります．図３・７・３にはフローチャートを示します．

1）このルーチンは，キー入力バッファ中から，ＫＨＥＡＤで示される場所のキーコードをＡレジスタに読み取ります．ただし，入力待ちしないときで入力がないときは，ＢレジスタとＡレジスタはクリアされます（ＫＥＹＩＮ０では，Ａレジスタは元の値のままです）．

2）ＫＨＥＡＤが，バッファの終り（＄Ｄ３７Ｆ）まで来た場合は，バッファの先頭（＄Ｄ３６０）に再設定されます（ＩＮＣＫＢＸにて），これは，キー入

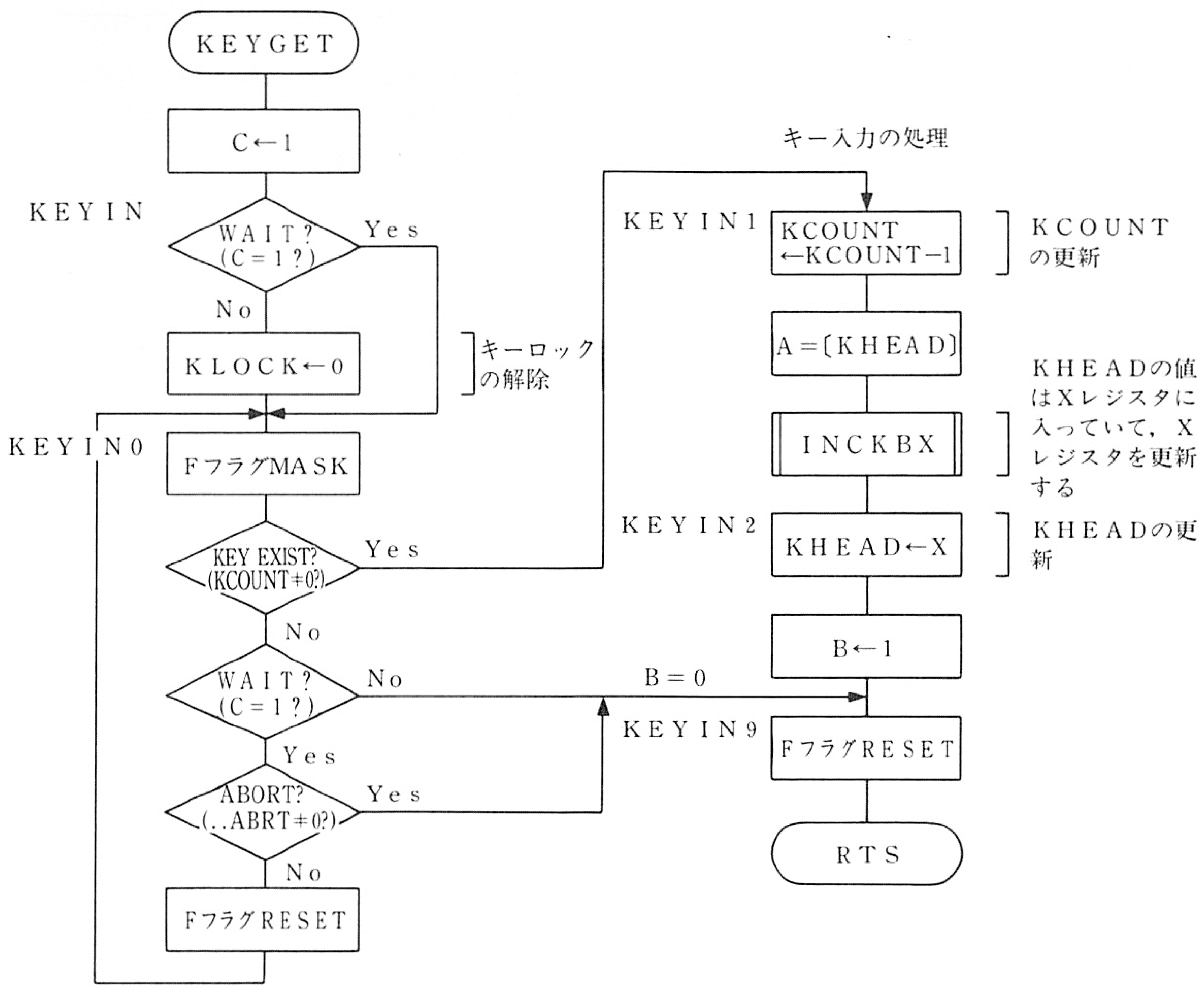
力バッファが循環して使用されるためです.

図3・7・2　バッファモードにおけるキー入力



※色のついている部分が，まだ読み込まれていないキーコードです．実際にはAレジスタおよびキー入力バッファには，その文字のアスキーコードが格納されています．

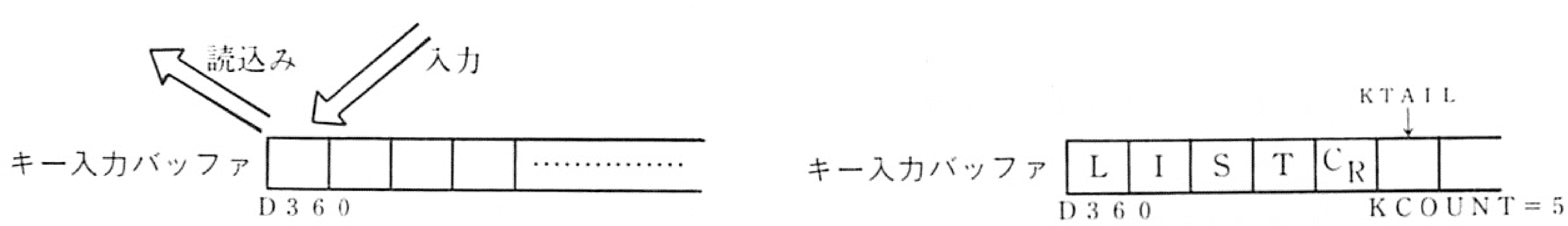
図3・7・3　KEYINの処理



キー入力処理は，ＦＩＲＱ割込みで行われているため，キーバッファからキーコードを読み込む部分や条件判断を行う部分は，Ｆフラグをマスクして FＩＲＱを禁止した状態で行われます（マスクしないと，読込みの時点でＦＩＲＱ割込みが起こってしまい，そちらの処理が移ると，ワークエリアの値が狂ってしまいます）．

バッファモードでないときは，ＫＥＹＩＲＱでいつもバッファを初期化（ＫＢＦＥＲＡを用います）するために，バッファの先頭の１バイト（＄Ｄ３６０）のみが使用されることになります(他の３１バイトは使用されません)．ただし，ＰＦキーについては，一度ＰＦキーを押したままで他のキーを押さないときは，順次バッファから読み込むことができます．その理由は，ＰＦキーの場合は，バッファモードでなくても，ＰＦキーの内容がＰＦキー定義テーブルからバッファへ一括して転送されるために，バッファ内に消えないで残る（ポインタ，カウンタが初期化されない）からです．

図3・7・4　ノンバッファモード



バッファモードでないときは，＄Ｄ３６０のみが用いられます．読込みをしないで入力が行われると前のデータが消えます．

ＰＦキーのときは，一括して転送されるため，他のキーを押さない限り，ポインタ・カウンタが初期化されないで残ります．

ＩＮＣＫＢＸ

＄ＦＤ９５～＄ＦＤ９Ｄ

機　能　ポインタの値を補正します．ポインタの値が終了番地を越えていたら，バッファの先頭アドレスに再設定します(キー入力バッファが循環している．つまり，リングバッファになっているためです)．

レジスタ　Ｘ

入力情報　Ｘレジスタ　ポインタ（ＫＨＥＡＤ）の値

出力情報　Ｘレジスタ　入力情報で与えられたポインタの値が＄Ｄ３８０に等しい場合は，＄Ｄ３６０を返します．それ以外のときは，入力情報のＸレジスタの値のままです．

ＫＢＦＥＲＡ

＄ＦＤ９Ｅ～＄ＦＤＡＢ

機　能　キー入力バッファをクリアします(実際にクリアするわけではなく，ポインタ，カウンタなどのワークエリアの内容を初期設定します)．

レジスタ　Ｘ

ワークエリア　ＫＣＯＵＮＴ　＄Ｄ００４　キー入力数
ＫＨＥＡＤ　＄Ｄ００５～＄Ｄ００６　キー入力バッファ読取り用ポインタ
ＫＴＡＩＬ　＄Ｄ００７～＄Ｄ００８　キー入力バッファバッファ書込み用ポインタ

解　説　ワークエリアを次のように初期化します．

ＫＣＯＵＮＴ　←　＄００
ＫＨＥＡＤ　←　＄Ｄ３６０（キー入力バッファの先頭アドレス）
ＫＴＡＩＬ　←　＄Ｄ３６０（キー入力バッファの先頭アドレス）

このようにすると，カウンタが0になっているので，まだ何もキー入力されていないのと同じになります．以後に入力されたキーコードは＄Ｄ３６０以降に蓄えられることになります．なお，ワークエリアを設定している間は，Ｆフラグをマスクして，ＦＩＲＱ割込み禁止しています．

ＫＥＹＩＲＱ

＄ＦＤＡＣ～＄ＦＤＥ５

機　能	ＦＩＲＱの割込み処理であるこのルーチンは，割込み発生させたキーインタフェースＬＳＩから，Ｉ／Ｏポートを通じてキーコードを受け取り，場合に応じてキーバッファに蓄えたり，割込みの制御を行ったりします．		
レジスタ	Ａ，Ｂ，Ｘ，Ｕ（ＣＣ，Ａ，Ｂ，Ｘ，Ｕは保存）		
ワークエリア	ＫＢＦＦＬＧ	＄Ｄ００３	キーバッファフラグ
	．ＰＦＩＲＱ	＄Ｄ００９	ＰＦキー割込み番号
	ＫＣＯＤ１	＄Ｄ４００	
	ＫＣＯＤ２	＄Ｄ４０１	
サブルーチン	ＫＢＦＥＲＡ	＄ＦＤ９Ｅ	キー入力バッファの初期化
	ＫＳＴＢＵＦ	＄ＦＤＥ６	キーコードをキー入力バッファへ格納
	ＦＵＮＡＤＲ	＄ＦＥ３Ｄ	ファンクションキー定義テーブルの計算

解　説　ＰＦキーと普通のキーでは，次のように動作が異ってきます．

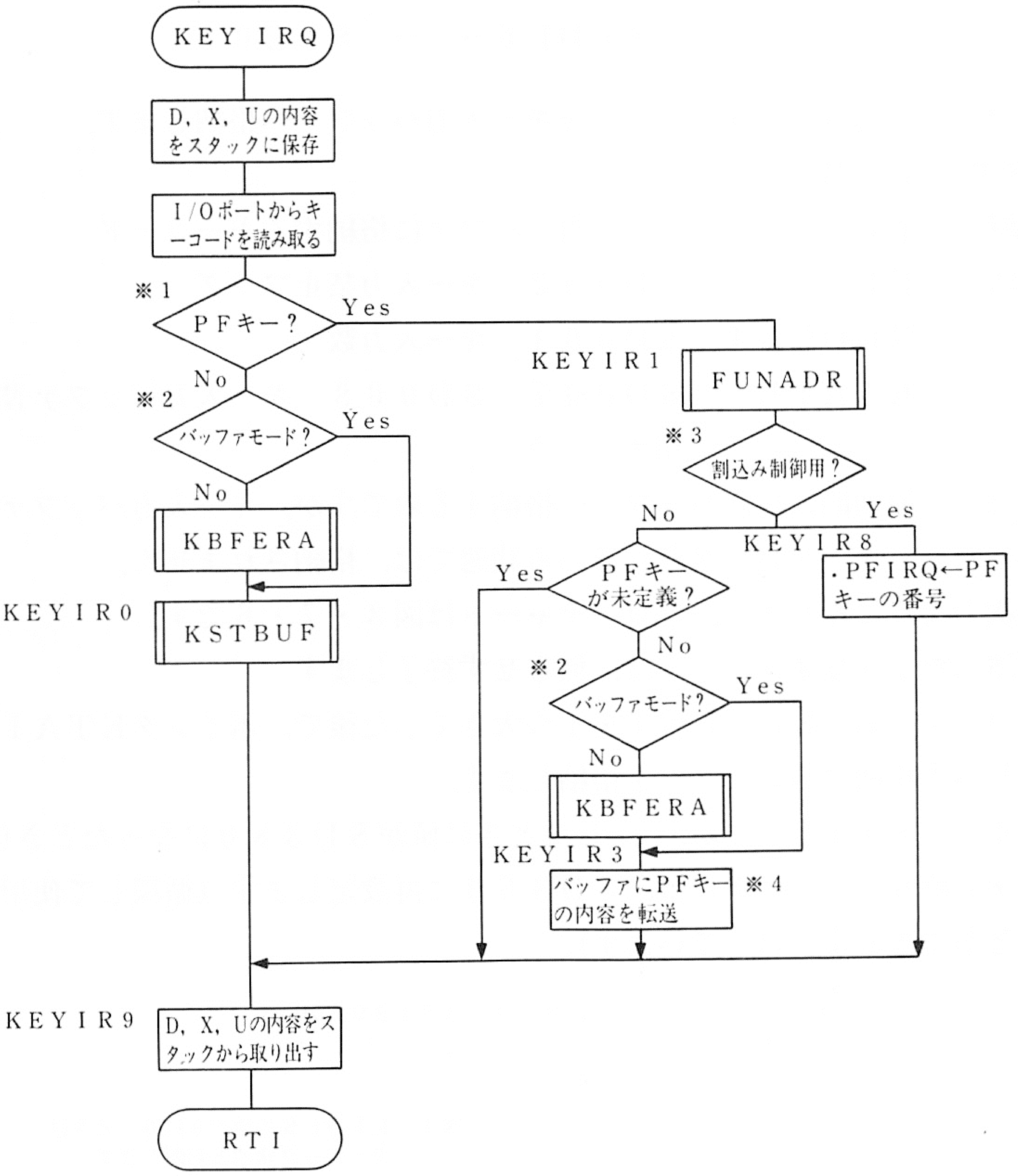
１）普通のキーの場合は，キーコードを入力バッファに蓄えていきます（バッファモードでないときは，最後に押したキーのコードのみが格納されることになります）．

２）ＰＦキーの場合は，

　ｉ）押されたＰＦキーが割込みの制御用に用いられている場合は，ワークエリア．ＰＦＩＲＱに割込みの起こったＰＦキーの番号を格納します（このワークエリアは，ＮＭＩであるＴＩＭＥＲ割込みの処理ルーチンによって参照され，指定があればメイン側に送られます）．

　ii）割込みの制御用でない場合は，ＰＦキー定義テーブルの内容をキー入力バッファに転送していきます(定義されていないときは，何もしません)．このとき，定義されている内容が一括して転送されるために，バッファモードでないときでも，最初の一文字以降のキーコードも消去されないで残ることになります．ただし，それらのコードが読み取られる前に他のキーを押すと，残りの部分も消去されることになります．

図3・7・5・　KEYIRQの処理

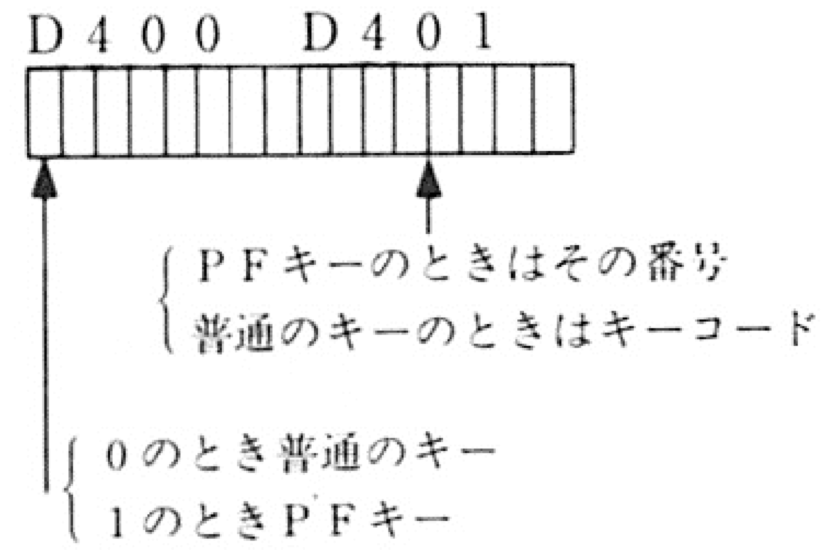


※1　PFキーかどうかの判別は，D400のビット7の内容を見ている．

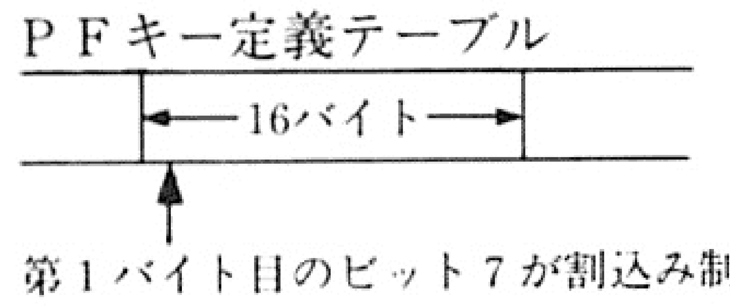
※2　バッファモードであるかどうかの判別は，ワークエリアKBFFLGを参照している．バッファモードでないときは，KBFERAでバッファを初期化してから蓄えていく

※3　割込み制御用かどうかは，そのPFキーの定義テーブルの第1バイト目のビット7を参照している．未定義かどうかは，ビット0～3を参照して，0であれば未定義とする．

※4　KSTBUFを用いている．



PFキー定義テーブル



第1バイト目のビット7が割込み制御用フラグでビット0～3が定義されているコードの長さ

ＫＳＴＢＵＦ

＄ＦＤＥ６～　～　＄ＦＥ０１

機　能	入力されたキーコードをキー入力バッファへ格納します．
レジスタ	Ａ，Ｂ，Ｕ
入力情報	Ｂレジスタ　　キー入力バッファに格納するキーコード
ワークエリア	ＫＬＯＣＫ　＄Ｄ００２　キー入力禁止フラグ ＫＣＯＵＮＴ　＄Ｄ００４　キー入力数 ＫＴＡＩＬ　＄Ｄ００７～＄Ｄ００８　キー入力バッファ書込み用ポインタ

解　説　原則的には，バッファへ格納するのですが，キー入力バッファが一杯であったり，キーがロックされている状態では，格納されません．

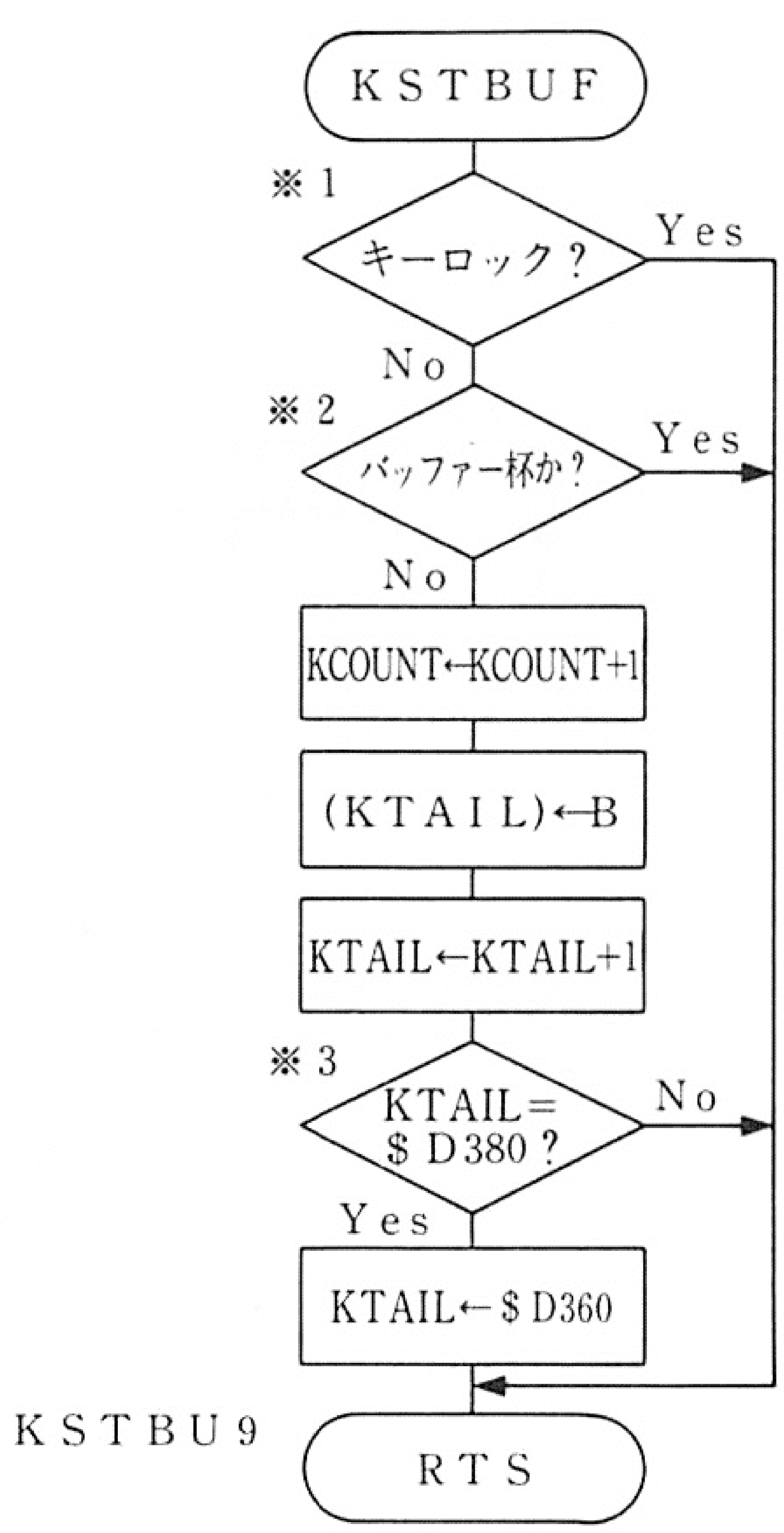
動作は次のようになります．フローチャートは図３・７・６です．

１）今述べたような条件のときは，何もせず終了します

２）カウンタであるＫＣＯＵＮＴを１つ大きくした後で，ポインタＫＴＡＩＬの指し示す場所にＢレジスタを格納します．

３）ポインタを＋１しますが，＋１したときに値が＄Ｄ３８０になったとき（バッファの終りにきたとき）は，＄Ｄ３６０に再設定します（循環して使用することができるようにしています）．

図３・７・６　ＫＳＴＢＵＦの処理



※１　ＫＬＯＣＫの値が０以外のときはキーロック状態と判断します

※２　ＫＣＯＵＮＴの値が＄２０であるときに，バッファが一杯であると判断します

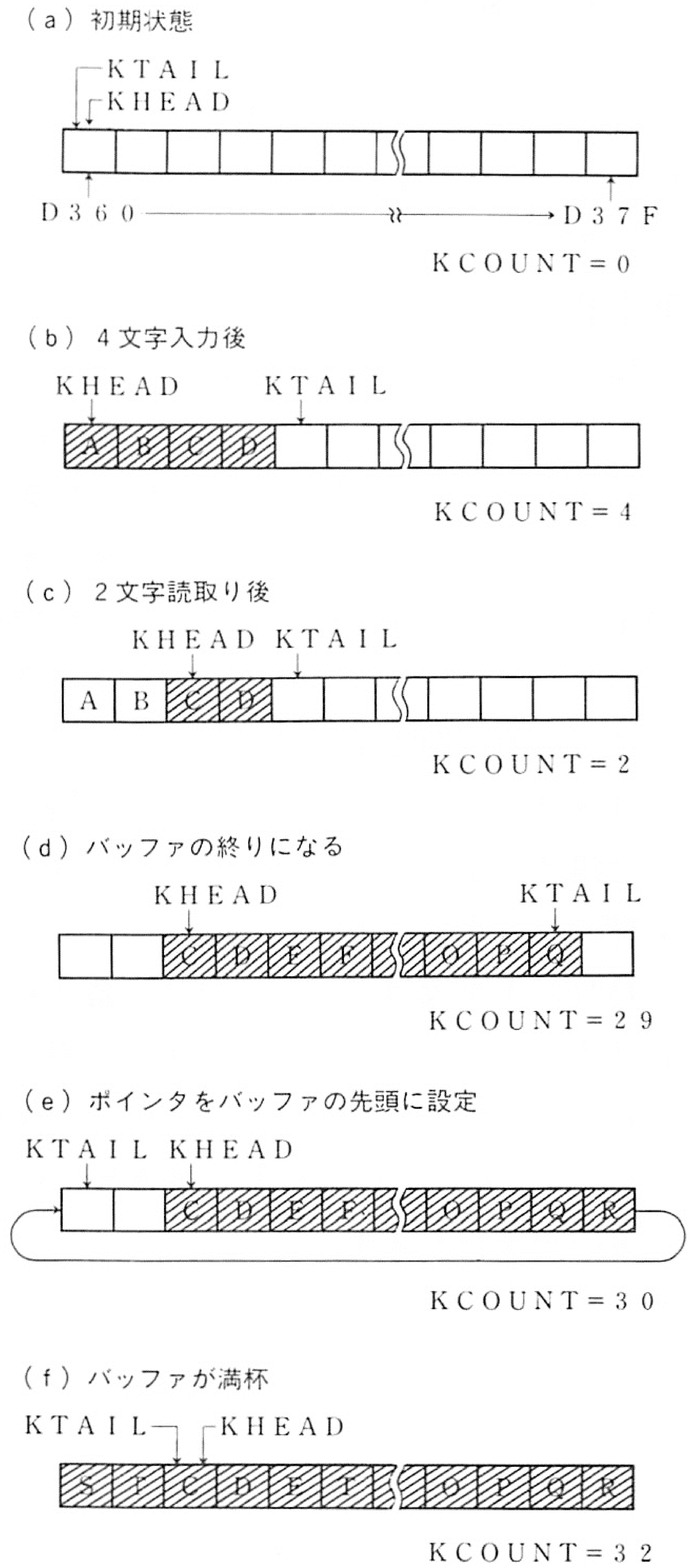
※３　実際には，ポインタの値はＵレジスタに代入されていて，更新・比較の対象はＵレジスタになります

キー入力バッファの循環構造（リングバッファ）について

キーが入力されると，KEYIRQにおいてKSTBUFを用いて，キー入力バッファにキーコードが格納されます．それらの格納されたキーコードは，KEYINが用いられると，キー入力された順から読み込まれていきます．このようにキー入力バッファは，先に入れたものから取り出していく形のFIFOバッファ(first in first out buffer)です．このため，バッファへコードを格納するためのポインタとバッファからコードを取り出してくるためのポインタとが必要になってきます．KTAILは格納用，KHEADは取出し用のポインタになっています．

2つのポインタは，初期状態では両方共＄D360に設定されています(a)．キーが入力されると，KTAILの指す場所にキーコードが格納されて，KTAILは＋1されていきます．また，まだ取り出されていないキーコードの数を示すカウンタKCOUNTも＋1されていきます(b)．また，KEYINでキーコー

図3・7・7　キー入力バッファ

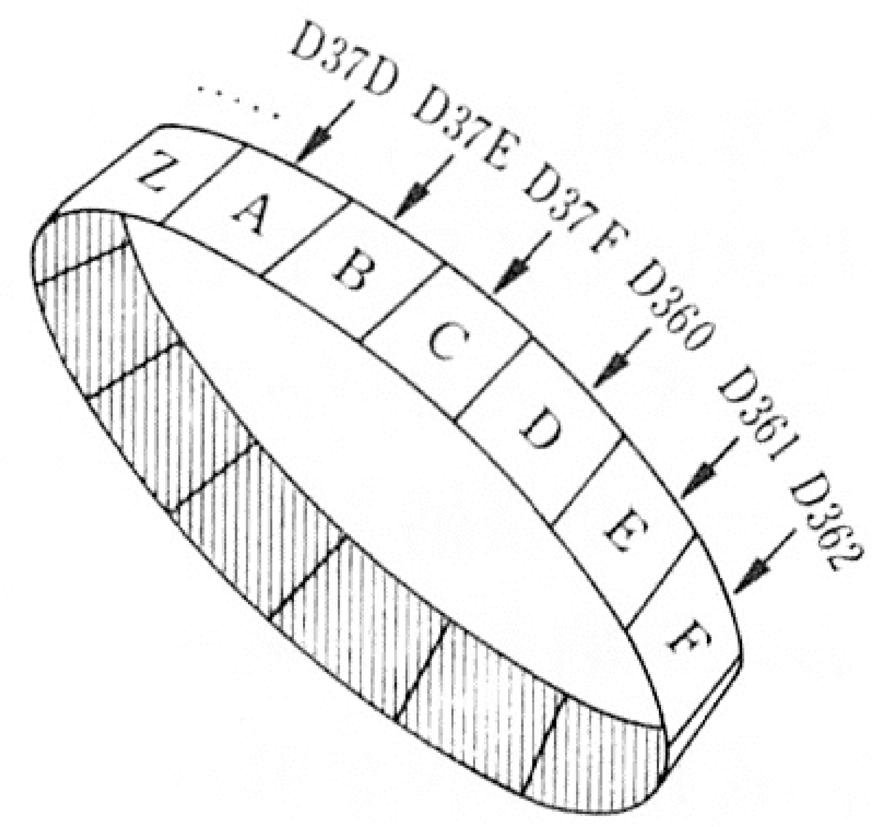


※KHEADも，バッファの終りに来たら次のコードを読み込んだ後は，バッファの先頭に設定されます．
色のついているものが，まだ読み込まれていないキーコードです．

ドが読み込むのは，ＫＨＥＡＤの示す場所から行います．ＫＨＥＡＤは読み込まれるたび＋１され，ＫＣＯＵＮＴは－１されていきます(c)．

ここで，キーを読み込まないでキー入力を続けていくと，バッファの終りになります(d)．ＦＩＦＯバッファですから，次にキーコードを格納した後は，バッファの先頭の空いている場所にポインタＫＴＡＩＬを設定すれば，効率よくバッファを使えることになります(e)．

図３・７・８　リングバッファ



キー入力バッファは循環しています

このように，ポインタがバッファの末尾にきたら，次のバッファの先頭に設定するという操作によって，バッファが循環しているようにみせています．循環させて使うバッファのことをリングバッファと呼ぶことがありますが，これはその一例です．

リングバッファにおいて，取出し用のポインタと格納用のポインタが一致した場所を指すのは，バッファ中にコードが何もないとき(a)か，バッファが一杯になったとき(f)のどちらかになります．このときのために，カウンタＫＣＯＵＮＴを設けて，バッファに蓄えられているキーコードの数を記憶しているのです．キー入力の際，キー入力バッファが一杯になっているとき（ＫＣＯＵＮＴ＝＄２０のとき）は，そのキー入力を無視します．なお，バッファモードを選択するには，Ｆ－ＢＡＳＩＣでＰＲＩＮＴ ＣＨＲ＄(＆Ｈ１Ｂ)＋“ｇ”を実行します．

3－7－2　Define String of ＰＦ　＄２Ａ　　（＄ＦＥ０２）（ＦＵＮＫＥＹ）

ＰＦキーに文字列を定義します．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　　容	値
Ｄ３８３	ＮＯ	ＰＦキー番号	１～10
Ｄ３８４	Ｎ	文字数	０～15
Ｄ３８５ ～ Ｄ３８４＋Ｎ	String	定義文字列	文字(のコード)列

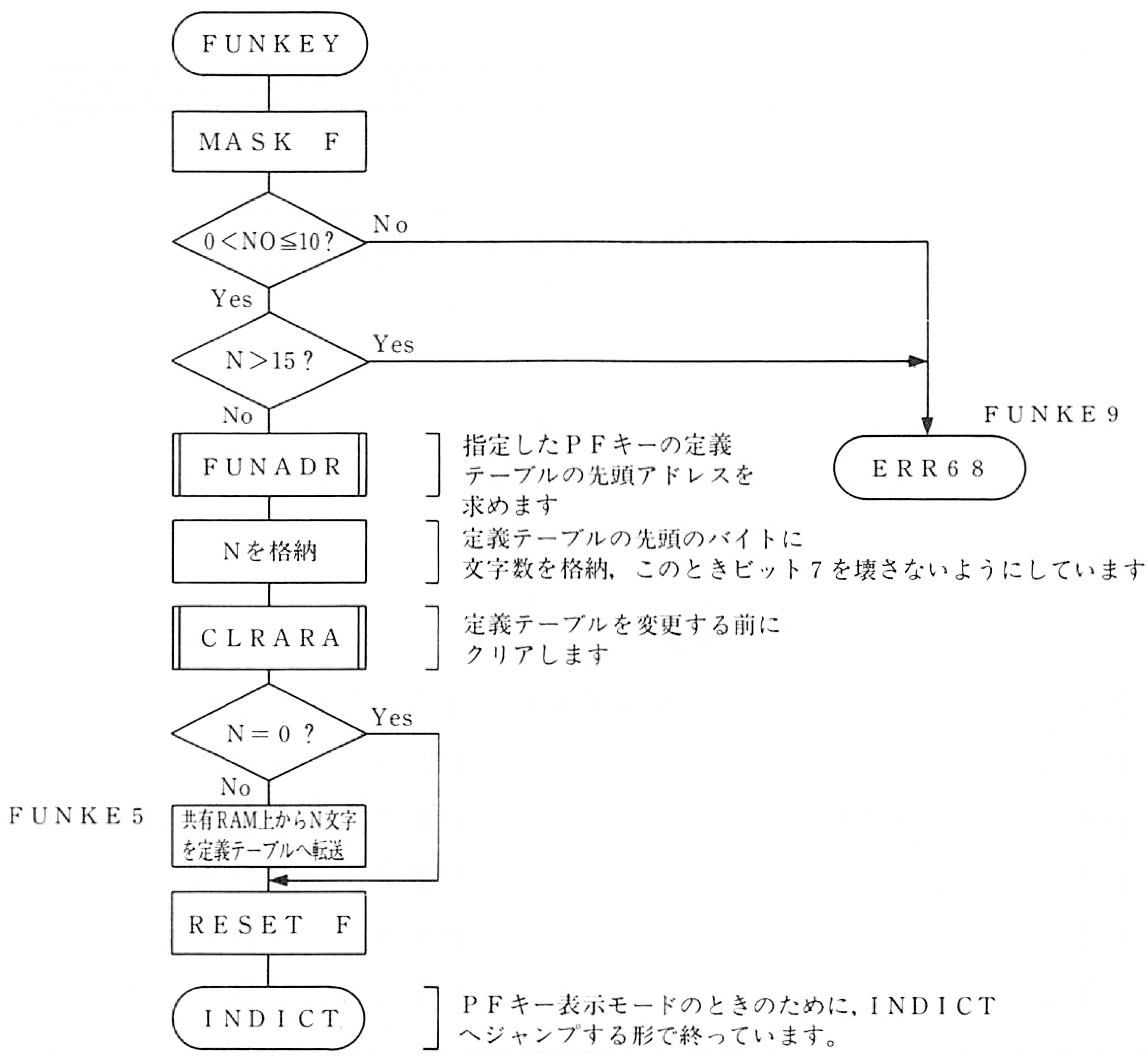
主なサブルーチン

＄ＦＥ３４　ＣＬＲＡＲＡ　特定領域をクリアする

＄ＦＥ３Ｄ　ＦＵＮＡＤＲ　ＰＦキー定義テーブル内の先頭アドレス検索

＄ＥＥ０Ｅ　ＩＮＤＩＣＴ　ＰＦキーの表示

図３・７・９　ＦＵＮＫＥＹの処理

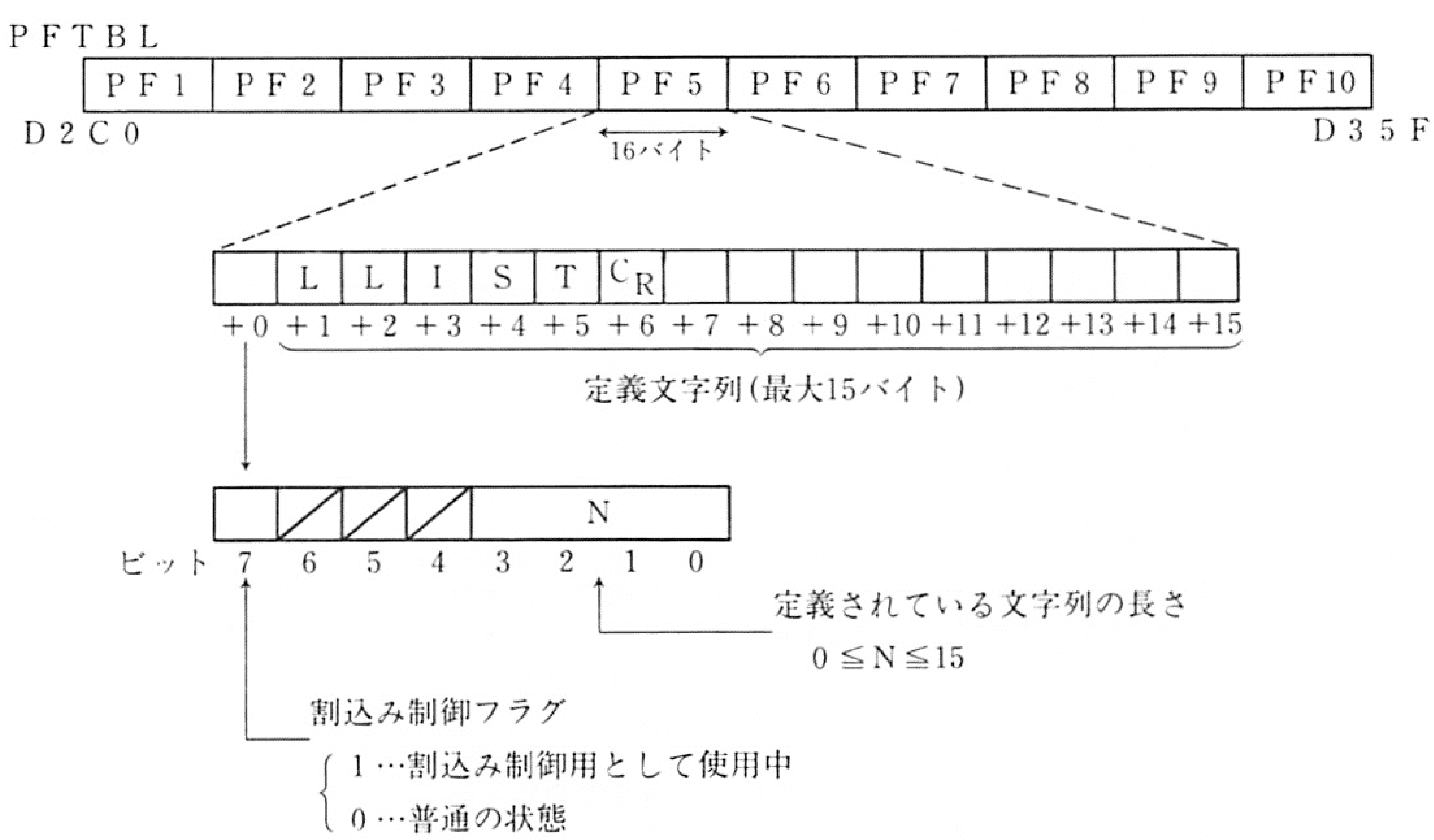


解　説　NOの示すPFキーの定義テーブルの先頭アドレスを求め，その定義テーブルをクリアした後，Nの示す文字数分だけ，共有RAM上からテーブルに定義文字列を読み込みます．この動作の間は，FフラグをMASKとしてFIRQを禁止しています．また，定義テーブルの先頭のバイトのMSB(ビット7)は，そのPFキーが割込み制御用かどうかを示すフラグになっていますが，このフラグを壊さないようにしています．このため，あるPFキーを割込み制御に使用しているときでも，PFキーの定義文字列を変更することができます．

PFキー定義テーブルについて

$D2C0〜$D35Fの領域はPFキーの定義文字列が格納されています．PF1〜10の各PFキーに対して16バイトずつ割り当てています．1つのPFキー定義テーブルの16バイト中の先頭の1バイトは，定義文字列の長さと割込み制御フラグとが格納されています（図3・7・10）．

図3・7・10　PFキー定義テーブル



各PFキーの定義テーブルの先頭アドレスは次のとおりです．

PF1	$D2C0	PF6	$D310
PF2	$D2D0	PF7	$D320
PF3	$D2E0	PF8	$D330
PF4	$D2F0	PF9	$D340
PF5	$D300	PF10	$D350

ＣＬＲＡＲＡ

＄ＦＥ３４　～　＄ＦＥ３Ｃ

機　能	特定の領域（大きさは２５６バイト以下）をクリアします．
レジスタ	Ｂ，Ｘ　（Ｘは保存）
入力情報	Ｂレジスタ　クリアする領域の大きさ（１以上２５６以下．ただし＄００は２５６バイトと見なされます） Ｘレジスタ　クリアする領域の先頭アドレス

解　説　このルーチンは，本来はＰＦキーの定義の際に，ＰＦキー定義テーブルを，再設定する前にクリアするために用いられています．

ＦＵＮＡＤＲ

＄ＦＥ３Ｄ　～　＄ＦＥ４５

機　能	Ｂレジスタの示す番号のＰＦキーの定義テーブルの先頭アドレスを求めます．
レジスタ	Ｂ，Ｘ
入力情報	Ｂレジスタ　ＰＦキーの番号（１～10）
出力情報	Ｘレジスタ　指定したＰＦキーの定義テーブルの先頭アドレス

解　説　次のような計算方法によって，Ｘレジスタを設定しています．

Ｘレジスタ＝＄Ｄ２Ｂ０＋Ｂレジスタ×16

３－７－３ Get String of ＰＦ ＄２Ｂ （＄ＦＥ４６）
（ＧＳＴＲＰＦ）

ＰＦキーに定義されている文字列を読み込み，共有ＲＡＭ上へ返します．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　　容	値
Ｄ３８３	ＮＯ	ＰＦキー番号	１～１０

共有ＲＡＭ上の復帰情報

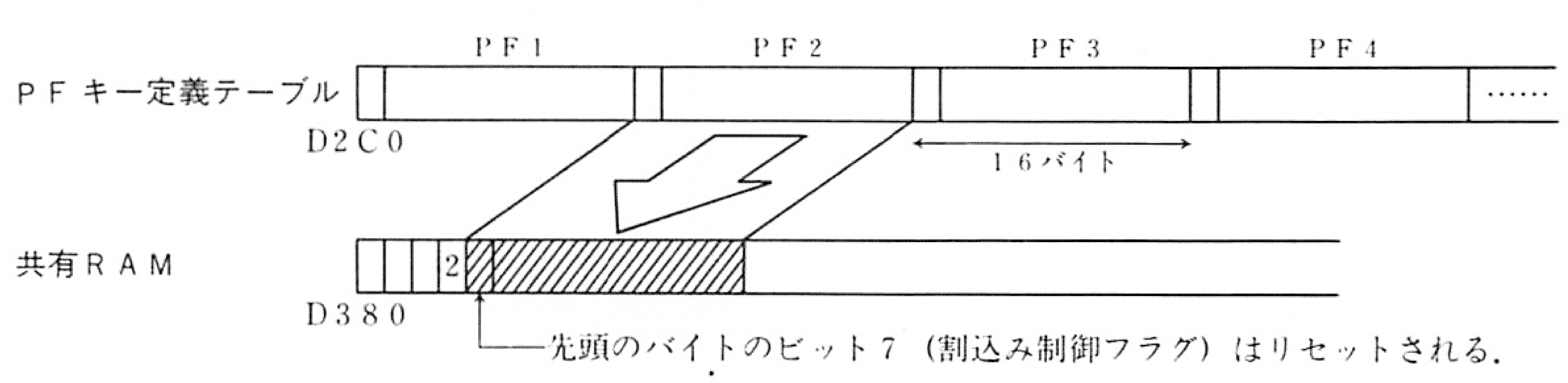
アドレス	記　号	内　　容	値
Ｄ３８３	ＮＯ	ＰＦキー番号	１～10
Ｄ３８４	Ｎ	文字数	０～15
Ｄ３８４ ～ Ｄ３８３＋Ｎ	String	定義されている文字列	文字列

主なサブルーチン

＄ＦＥ３Ｄ　　ＦＵＮＡＤＲ　ＰＦキー定義テーブルの先頭アドレスの検索

解　説　ＦＵＮＡＤＲを用いて，ＮＯの示す番号のＰＦキーの定義テーブルの先頭アドレスを求めた後，そこから16バイトを共有ＲＡＭ上（＄Ｄ３８４～＄Ｄ３９４）へ転送します．ただし，先頭のバイトのビット７は割込み制御フラグなのでビット７をリセットしてから，先頭の１バイトを共有ＲＡＭ上に送っています．

図3・7・11　定義テーブルの内容を共有ＲＡＭへ転送



3－7－4　Interrupt　Control　＄２Ｃ　（＄ＦＥ６Ｉ）（ＰＦＩＮＴ）

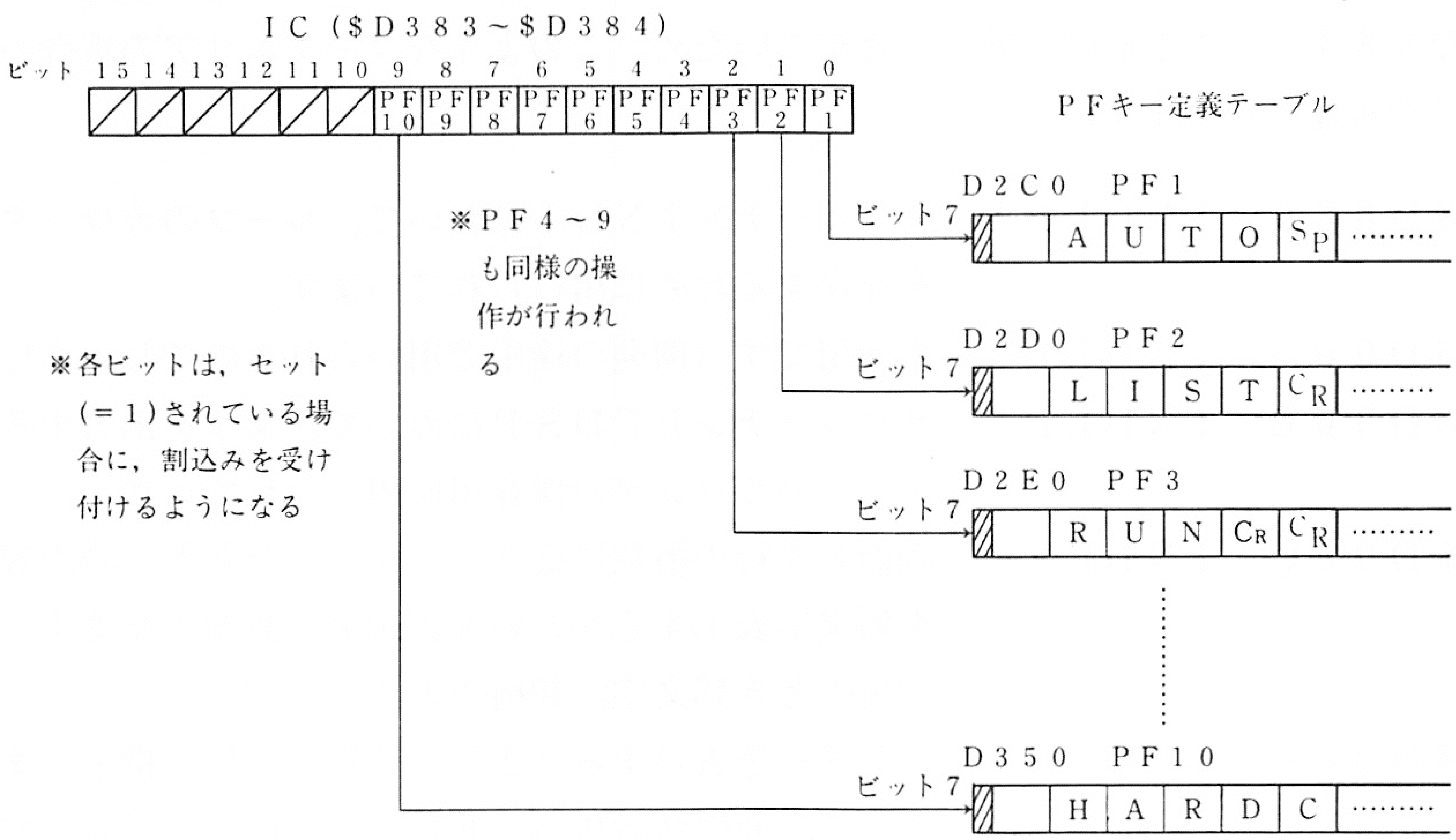
ＰＦキーによる割込みの選択を行います．

共有ＲＡＭ上のパラメータ

アドレス	記　号	内　容	値
Ｄ３８３〜Ｄ３８４	ＩＣ	割込み制御フラグ(16ビット)	１で割込みイネーブル

解　説　共有ＲＡＭ上のＩＣを読み込み，それに従って各ＰＦキー定義テーブルの先頭の１バイトのビット７をセット（クリア）していきます．セットしている間はＦフラグをマスクして，キー入力によるＦＩＲＱが起こらないようにしています．

図3・7・12　割込み制御フラグの設定



ＰＦＩＮＤＣ（＄ＦＥ７７）からはＦフラグをリセットしてＦＩＲＱを許可した後で，ＩＮＤＩＣＴ（＄ＥＥ０Ｅ）へジャンプしてＰＦキーを表示する（表示モードのとき）という動作を行っています．

ＩＮＤＩＣＴ

＄ＥＥ0Ｅ　～　＄ＥＥ３２

機　　能	ＰＦキー表示モードのとき，画面の下２行にＰＦキーを表示します．表示モードでないときは何もしません．		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
ワークエリア	..ＢＣＬ	＄Ｄ０２Ｆ	バックカラー
	..ＣＷＤ	＄Ｄ０３１	キャラクタの幅
	..ＦＳ	＄Ｄ０３８	ＰＦキー表示フラグ
	..DMAP+6	＄Ｄ０４３	（ＰＦキー表示キー表示画面開始行）
	解説参照	＄Ｄ０６９～＄Ｄ０６Ｄ	
サブルーチン	ＩＮＤＩ	＄ＥＥ３３	ＰＦキー１行表示ルーチン

解　　説　ＩＮＤＩＣＴ処理系のサブルーチン群（ＩＮＤＩ，ＩＮＤＣＨＲ，ＰＦＤＳＰ）において，ローカル（局所的）なワークエリアの使用が行われます．そのワークエリアの領域は＄Ｄ０６９～＄Ｄ０６Ｄで，普通は．ＢＯＸＸ１～．ＢＯＸＸ２に用いられていますが，ＰＦキー表示の際は下記のような使われ方になります（両方を同時に使うことがないため，このようなワークエリアの重合わせが可能となります）．

＄Ｄ０６９	ＩＮＤＬＰ１	サブルーチンＩＮＤＩにおいて，ループのカウンタを保存するために用いられています．
＄Ｄ０６Ａ	ＩＮＤＬＰ２	未使用です（開発の途中で用いられたのでしょう）．
＄Ｄ０６Ｂ	ＩＮＤＬＰ３	サブルーチンＰＦＤＳＰにおいて，文字を出力するループのカウンタの保存用に用いられています．
＄Ｄ０６Ｃ	ＩＮＤＰＦ	画面の１行の桁数に応じて，１つのＰＦキーの内容を何文字表示するかという表示文字数が入ります．80桁のとき15文字，40桁のとき７文字．
＄Ｄ０６Ｄ	ＩＮＤＢＳＶ	ＰＦキーを表示するときに，背景色を黒に指定しますので，元の背景色（..ＢＣＬ）を保存しておくために用いられています．

ＩＮＤＩＣＴの処理系の各サブルーチンの役割は，次のようになります．

ＩＮＤＩＣＴ　各サブルーチンが参照するパラメータの設定，メインルーチン．

ＩＮＤＩ　　　ＰＦＤＳＰを用いて，５個（１行）ＰＦキーを出力する．

ＰＦＤＳＰ　　１つのＰＦキーの内容を表示する．
ＩＮＤＣＨＲ　ＰＦキーの内容の１文字を表示する．

ＩＮＤＩＣＴの動作は次の通りです．

1）ＰＦキー表示モードが判別し，表示モードでなければ(..ＦＳ＝０)，終了します．

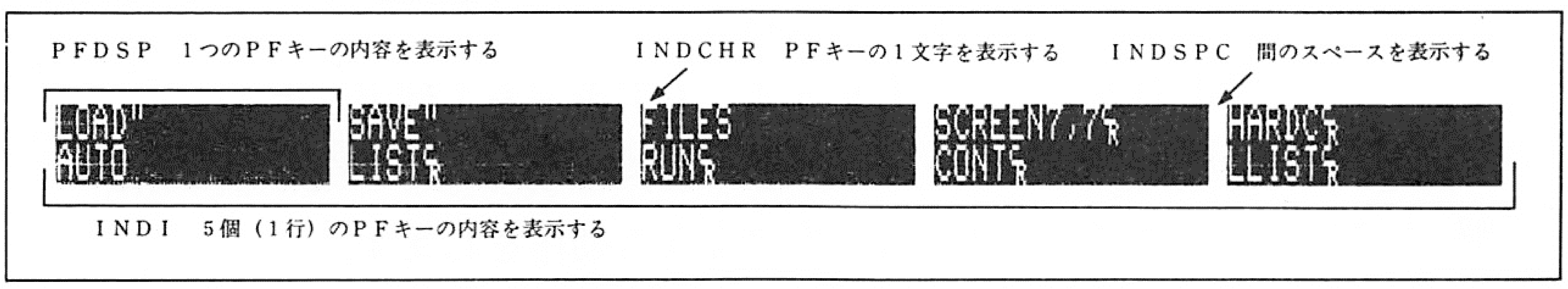
2）画面の１行が80桁か40桁か判別し，40桁ならＩＮＤＰＦに７を80桁なら15を代入して，１つのＰＦキーの表示桁数を決めます．

3）背景色（..ＢＣＬ）の値を，ＩＮＤＢＳＶに保存した後，..ＢＣＬをクリアして背景色を黒にします．

4）ＰＦキー表示画面（２行）の下の行に，ＰＦ１～５の内容を，上の行にＰＦ６～１０の内容をＩＮＤＩを使用して表示します．

5）ＩＮＤＢＳＶに保存しておいた元の背景色を再び..ＢＣＬに戻します．

ＩＮＤＩＣＴ　すべてのＰＦキーの内容を表示する



ＩＮＤＩ

＄ＥＥ３３　～　＄ＥＥ４２

機　能	指定された行へＰＦキーの内容を５個表示します．	
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｚ	
入力情報	Ａレジスタ	ＰＦキーの内容を出力する行（０～24）
	Ｘレジスタ	ＰＦキー定義テーブルの先頭アドレス(＄Ｄ２Ｃ０，＄Ｄ２Ｄ０，＄Ｄ２Ｅ０，＄Ｄ２Ｆ０，＄Ｄ３００，＄Ｄ３１０のいずれか)
	ＩＮＤＰＦ	＄Ｄ０６Ｃ　１つのＰＦキーの表示文字数（≠15）
出力情報	Ｘレジスタ	最終に表示したＰＦキーの次のＰＦキーの定義テーブルの先頭アドレス
ワークエリア	ＩＮＤＬＰ１	＄Ｄ０６９
サブルーチン	ＢＶＡＤＬ	＄Ｅ２２１　指定した行の先頭のＶＲＡＭ上のアドレスを求める
	ＰＦＤＳＰ	＄ＥＥ５４　ＰＦキーを１個表示する

解　説　このサブルーチンは，次のような動作をします．

１）与えられたＡレジスタを基に，ＢＶＡＤＬを用いてＶＲＡＭ上のアドレスを求め，Ｕレジスタに代入しています．

２）ＰＦＤＳＰを５回呼び出して，５個のＰＦキーを表示させます．このループのカウンタとしてＩＮＤＬＰ１を用います．

ＰＦＤＳＰを呼び出した後のＸレジスタはＰＦＤＳＰによって自動的に次のＰＦキー定義テーブルの先頭アドレスに設定されていますから，そのままＰＦＤＳＰを繰り返して呼び出すだけでよいのです．ＶＲＡＭ上のアドレスを示すＵレジスタに関しても同じことが言えます．

ＩＮＤＰＦに設定しておく値は，１行の桁数が80桁のときは最大15，40桁のときは，最大７までとなります．この値以上になると重ね書きされて見にくくなります．

次の例は，キャラクタＹ座標１の行へ，ＰＦ１～５の内容を５文字ずつ表示していきます．

```
LDA  #5
STA  INDPF
LDA  #1
LDX  #$D2C0
JSR  INDI
```

ＰＦＤＳＰ

＄ＥＥ５４　～　＄ＥＥ７１

機　能	１つのＰＦキーの内容を表示します		
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	Ｘレジスタ	表示するＰＦキーの定義テーブルの先頭アドレス	
	Ｕレジスタ	ＰＦキーを表示するＶＲＡＭ上のアドレス	
	ＩＮＤＰＦ	＄Ｄ０６Ｃ	ＰＦキーの表示文字数（≦15）
出力情報	Ｘレジスタ	表示したＰＦキーの次のＰＦキーの定義テーブルの先頭アドレス	
	Ｕレジスタ	次のＰＦキーを表示すべきＶＲＡＭ上のアドレス	
ワークエリア	ＩＮＤＬＰ３	＄Ｄ０６Ｂ	
	ＩＮＤＰＦ	＄Ｄ０６Ｃ	
サブルーチン	ＩＮＤＳＰＣ	＄ＥＥ４３	スペースを表示する
	ＩＮＤＣＨＲ	＄ＥＥ４５	１文字表示する

解　説　動作は次のようになります

1）定義テーブルの先頭の１バイトのビット７を見て，表示するＰＦキーが割込み制御用なら文字色を１１（紫の反転表示）にします．制御用でなけれな文字色を１３（水色の反転表示）にします．

2）定義テーブルの２バイト目からの内容を，ＩＮＤＣＨＲを用いて表示文字数分だけ出力します．内容の表示はＩＮＤＰＦの値でＩＮＤＬＰ３を設定し，一定表示する度にＩＮＤＬＰ３を－１していき，０になるまで行います．

3）Ｘレジスタを入力値に１６（１つのＰＦキーテーブルのサイズ分だけ）足して返します．最後にＩＮＤＳＰＣを使って，次のＰＦキーとの間のスペースを１個出力して終了します．

次の例は，ＶＲＡＭの＄２０００番地にＰＦ３の内容を９文字表示します．

```
LDA  #9
STA  INDPF
LDX  #$D2E0
LDU  #$2000
JSR  PFDSP
```

ＩＮＤＣＨＲ

ＩＮＤＳＰＣ　　＄ＥＥ４３　～　＄ＥＥ５３
ＩＮＤＣＨＲ　　＄ＥＥ４５

機　能	１文字出力します．ＩＮＤＳＰＣは空白を１つ出力します．
レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ　（Ａ，Ｂ，Ｘは保存）
入力情報	Ａレジスタ　出力する文字のコード（ＩＮＤＣＨＲのみ）
	Ｂレジスタ　文字色（０～１５）　（ＩＮＤＣＨＲのみ）
	Ｕレジスタ　ＶＲＡＭ上のアドレス
出力情報	Ｕレジスタ　次の文字を出力するべき，ＶＲＡＭ上の位置
ワークエリア	..ＶＣＵ　＄Ｄ０３９　ＶＲＡＭ上の桁単位
サブルーチン	ＣＤＳＰ　＄Ｅ５１Ｄ　キャラクタを１文字出力

解　説　このルーチンは，ＰＦキーの表示ルーチン群の一部ですが，独自に文字を１文字出力するルーチンとして用いることができます．このルーチンを使うと，ＶＲＡＭ上のアドレスが自動的に１文字分だけ増やされて返されますので，続けて文字を出力するのに有効なルーチンです．
動作は次のようになります．

１）Ａ，Ｂ，Ｘ，Ｕレジスタをスタック上に退避します．ＸレジスタにＵレジスタの値を代入しておきます（ＣＤＳＰに受け渡すためです）．
２）ＣＤＳＰを呼び出して，キャラクタのフォントを出力します．
３）スタック上に退避してあるＵレジスタの値に..ＶＣＵの値を代入して，１文字分ＶＲＡＭ上にアドレスを進めます．
４）スタック上にあるレジスタをとり出して終了します．

ＩＮＤＳＰＣは，上記の動作に入る前に，ＡレジスタとＢレジスタを＄００に設定しています．

次の例は，ＶＲＡＭの＄２０００番地の座標に，黄色の反転表示で“ＦＭ”と書きます．

```
LDA   #'F
LDB   #14
LDU   #$2000
JSR   INDCHR
LDA   #'M
JSR   INDCHR
```

３－７－５　Set　Timer（ＳＥＴＴＩＭ）＄３Ｄ（＄ＦＥ７Ｃ）
Read　Timer（ＲＥＤＴＩＭ）＄３Ｅ（＄ＦＥＡ４）

各タイマレジスタの値の書込み（Set Timer）もしくは読込み（Read Timer）を行います．特に制御レジスタ．＄ＴＣ（＄Ｄ００Ｂ）への書込み設定を行った場合にはタイマ関係の割込みがアクティブとなり，指定された割込み事象が発生した場合にはメイン側に対してアテンション割込み要求がかかるようになります．

共有ＲＡＭ上のパラメータ（Set Timer）

アドレス	記　号	内　　　容	値
Ｄ３８３	ＲＳ	設定レジスタ選択フラグ	ビット０：ＴＣ ビット１：Ｔ１ ビット２：Ｔ１Ｉ ビット３：Ｔ２ ビット４：Ｔ２Ｄ
Ｄ３８４	ＴＣ	制御レジスタ	ビット０：予約時刻割込みＯＮ／ＯＦＦ ビット１：インタバルタイマ割込み 〃 ビット２：インタバルワンショット指定 ビット３：０時割込みのＯＮ／ＯＦＦ
Ｄ３８５ 〜 Ｄ３８８	Ｔ１	時刻レジスタ （クロック）	第１バイト：時刻 第２バイト：分 第３バイト：秒 第４バイト：20ms
Ｄ３８９ 〜 Ｄ３８Ｃ	Ｔ１Ｉ	割込み予約時刻レジスタ	第１バイト：時刻 第２バイト：分 第３バイト：秒 第４バイト：20ms
Ｄ３８Ｄ 〜 Ｄ３９０	Ｔ２	デクリメントカウンタ （1step ＝20ms） ※約２年８ヵ月と20日分	32ビット（２の32乗）×20msデクリメントカウンタデータ
Ｄ３９１ 〜 Ｄ３９４	Ｔ２Ｄ	カウンタ再設定値レジスタ	32ビットデータ

共有ＲＡＭ上の復帰情報（Read Timer）

アドレス	記　号	内　　　容	値
Ｄ３８３	―――	――――――	不　定
Ｄ３８４	ＴＣ	制御レジスタ	１バイトデータ(各ビットの意味はSet Timerと同様)
Ｄ３８５ 〜 Ｄ３８８	Ｔ１	時刻レジスタ	４バイトデータ(各バイトの意味はSet Timerと同様)
Ｄ３８９ 〜 Ｄ３８Ｃ	Ｔ１Ｉ	割込み予約時刻レジスタ	４バイトデータ(同上)
Ｄ３８Ｄ 〜 Ｄ３９０	Ｔ２	デクリメントカウンタレジスタ	３２ビットデータ
Ｄ３９１ 〜 Ｄ３８４	Ｔ２Ｄ	再設定値レジスタ	３２ビットデータ

主なサブルーチン

＄Ｅ４４Ａ　ＭＯＶＥ　ブロック転送ルーチン
＄ＦＥＢ２　ＴＭＣＴ　タイマ処理系エンディングルーチン
＄ＦＥＣ８　ＴＩＭＥＲ　タイマカウント実行ルーチン

主なワークエリア

各タイマレジスタ（＄Ｄ００Ｂ〜＄Ｄ０１Ｂの．＄ＴＣ〜．＄Ｔ２Ｄ)
＄Ｄ００Ａ　．ＴＭＡＣ　タイマアクセスフラグ

解　説　この処理系は20msごとにかかるＮＭＩ（ノンマスカブルインタラプト）と深い関係にあります．ＮＭＩがかかるとＴＩＭＮＭＩ（＄ＦＥＢＦ）というサービスルーチンに入って，各タイマレジスタのカウントアップ・ダウンを行うしくみになっておりますが，メイン側でこれらタイマレジスタ類の設定を行う場合，もしくはそれらを参照する場合に，それぞれＳＥＴＴＩＭ(Set Timer)コマンド，ＲＥＤＴＩＭ（Read Timer）コマンドが用いられるわけです．

ここで注意しなければならないのは，コマンド処理によるタイマレジスタ類のアクセスと，ＮＭＩによるタイマレジスタ類のカウントアップ（or ダウン）とが競合してしまう可能性があるという点です．例えば，12時59分59秒プラス980msの時点で，ＲＥＤＴＩＭコマンドによってレジスタの読取りを始めたといたします．

「12時」のところまで読取りを終了したところで，ＮＭＩがかかって時刻レジスタのカウンタアップが実行された場合，レジスタの値は「13時00分00秒プラス０ms」となるわけですが，すでに「12時」だけ読取ってしまっているため，「12時00分……」という不正確な値がメイン側に渡されることになります．

このような不都合を回避するためには，タイマレジスタ類のアクセスの競合を防ぐ必要があるのですが，タイマ割込みはＮＭＩによって生じるため，いかなるソフトウェア手段をもってしても，割込みの発生自体はマスクすることができません．そこでタイマアクセスフラグ．ＴＭＡＣ（＄Ｄ００Ａ）というフラグを設け，ＳＥＴＴＩＭ・ＲＥＤＴＩＭ処理系によってタイマレジスタ類がアクセスされている場合には，その旨を知させるためにフラグを反転して＄ＦＦとします（普通はフラグは＄００となっています）．もしもアクセス終了前にＮＭＩがかかってサービスルーチンに飛んだとしても，このフラグをチェックすることによってタイマレジスタのカウントアップ（ダウン）を控え，ＲＴＩ命令により即座に元の処理系に戻るようにしておけば，競合を防ぐことができるわけです．ただ，アクセスフラグが立っていたことによってカウントアップ（ダウン）が１回完全に無視されてしまうと，タイマの値が狂ってしまうため，ＲＴＩする直前にアクセスフラグをクリアして，ＮＭＩがかかったということを元の処理系に知らせるようにします．ＳＥＴＴＩＭ・ＲＥＤＴＩＭ両者のエンディングルーチンを兼ねるＴＭＣＴでは，タイマ設定・読込みの最中にＮＭＩがかかったかどうかをアクセスフラグを見ることによってチェックし，もしもＮＭＩがあった場合，すなわちアクセスフラグがクリアされていた場合には，一時的に控えていたタイマレジスタのカウントアップ（ダウン）を遅ればせながら実行するわけです．なおタイマレジスタのカウントを実行するのは，ＴＩＭＥＲ（＄ＦＥＣ８）という名前のサブルーチンです．

ＳＥＴＴＩＭでは，アクセスフラグを立てた後，共有ＲＡＭ上のパラメータＲＳを参照して，指定されたタイマレジスタのみ共有ＲＡＭからワークエリアへの転送を行い，設定を実行した後ＴＭＣＴに入ります．

一方ＲＥＤＴＩＭでは，アクセスフラグを立てた後，ワークエリア上のすべてのタイマレジスタを共有ＲＡＭ上に読み出しますが，この読出しにはＭＯＶＥルーチンを利用しております．読出し後はやはりＴＭＣＴに入ります．

ＴＭＣＴ

ＴＭＣＴ　　　　＄ＦＥＢ２　～　＄ＦＥＢＥ

機　　能　タイマアクセスフラグを調べ，タイマレジスタのアクセス処理の間にＮＭＩがかかった場合（アクセスフラグ＝＄００のとき）には，ＴＩＭＥＲサブルーチンを呼び出してタイマのカウントアップ（or ダウン）を実行します．

レジスタ　ＴＩＭＥＲサブルーチンを呼び出さない場合には使用せず，呼び出した場合にはＴＩＭＥＲに同じ．

入力情報　．ＴＭＡＣ　　＄Ｄ００Ａ　タイマアクセスフラグ

出力情報　．ＴＭＡＣ　　（処理後，クリアされて＄００となる）

サブルーチン　ＴＩＭＥＲ　　＄ＦＥＣ８　タイマカウント実行ルーチン

解　　説　前述のとおりタイマレジスタのアクセスの競合防止のために一時的にタイマカウントアップ（or ダウン）が控えられた場合に，タイマの値が狂わないように事後処置を行うルーチンです．

注目すべき点は，ＴＩＭＥＲルーチンを呼び出す前にＦフラグを立てることによってキーボードからの割込み(ＦＩＲＱ)をマスクしていることです．これはＴＩＭＥＲルーチンが，タイマ関係の割込みだけでなく，ＰＦキーからのアテンション割込みの発生をメイン側に対して実際に知らせる役目も一括して果たしていることによるのです．ＮＭＩサービスルーチンからＴＩＭＥＲが呼び出された場合には，ハードウェア的にＦフラグがマスクされているのですが，普通の状態ではＦマスクはクリアされている（割込みイネーブル状態）ため，このように人為的にマスクを行う必要があるわけです．ＴＭＣＴルーチンの最後で，再びＦマスクはクリアされます．

ＴＩＭＮＭＩ

ＴＩＭＮＭＩ　　　＄ＦＥＢＦ　～　＄ＦＥＣ７

機　能	20msタイマ用ＮＭＩ（ノンマスカブルインタラプト）のサービスルーチンです．前述のとおりタイマアクセスフラグが立っていない場合には，ＴＩＭＥＲサブルーチンを呼び出して，各タイマレジスタのカウントアップもしくはカウントダウンを行います．アクセスフラグが立っていた場合には，アクセスフラグをクリアしてからＲＴＩ命令で即座に元の処理系にリターンします．

レジスタ	Ａ（ＴＩＭＥＲを呼び出した場合のことはＴＩＭＥＲ参照）		
入力情報	．ＴＭＡＣ	＄Ｄ００Ａ	タイマアクセスフラグ
出力情報	．ＴＭＡＣ	（処理後，クリアされる）	
サブルーチン	ＴＩＭＥＲ	＄ＦＥＣ８	次の項の本文参照

ＴＩＭＥＲ

ＴＩＭＥＲ　　　＄ＦＥＣ８　～　＄ＦＦ６Ａ

機　能	各タイマのカウントアップもしくはカウントダウンを実行するとともに，カーソルの点滅やメイン側へのアテンション割込み要求などの制御も行います．

レジスタ	Ａ，Ｂ，Ｘ，Ｙ，Ｕ		
入力情報	．ＣＵＲＳＲ	＄Ｄ０１Ｃ	カーソル表示フラグ
	．ＰＦＩＲＱ	＄Ｄ００９	ＰＦキー割込み番号
出力情報	ＳＴＡＴＵＳ	＄Ｄ３８１	ステータスレジスタ
	ＡＴＴＥＮＴ	＄Ｄ４０４	アテンション割込み要求ポート
ワークエリア	．ＣＵＲＳＴ	＄Ｄ０１Ｅ	カーソル状態フラグ
	．ＣＵＲＴＭ	＄Ｄ０１Ｄ	カーソルタイマ
サブルーチン	ＣＤＳＰＣ	＄Ｅ５１８	キャラクタ表示ルーチン（カーソル用） ［Areg：文字コード，Breg：アトリビュート］

解　説　このルーチンの処理は次のような手順で行われます．

1）まずカーソル表示フラグ．ＣＵＲＳＲを見て，現時点でのカーソルの表示状態をチェックし，表示状態であれば（．ＣＵＲＳＲ＝＄００）カーソルタイマ．ＣＵＲＴＭをデクリメントして，０であるかどうかをテストします．もしもカーソルタイマ＝０の場合には，2)の反転処理を行います．一方カーソルが表

示状態でない場合（中止状態もしくは禁止状態：これらカーソル表示状態の区別については，３－４－５節Console Controlを参照してください)，およびカーソルタイマ≠０の場合には，3)の処理へと分岐します．

2）現在のカーソル位置の文字コードおよびアトリビュートコードを取り出して（この際，．ＣＢＡＤおよび．ＣＡＡＤを間接参照します）ＡおよびＢレジスタに入れ，カーソル状態フラグ．ＣＵＲＳＴを反転します．反転の結果その値が＄ＦＦだった場合にはBレジスタ内のアトリビュートコード第3ビット（リバース指定）を反転してから，値が＄００だった場合にはリバース指定の反転を行わずにＣＤＳＰＣルーチンを呼び出して，カーソル位置の文字を表示します．つまり．ＣＵＲＳＴが＄００のときは，カーソル位置の文字は正常表示され，＄ＦＦのときには反転表示（ノーマル指定のときにはリバース表示，リバース指定の場合にはノーマル表示）されるわけです．同時にカーソルタイマ．ＣＵＲＴＭに値２０を代入します．ＮＭＩがかかるのは２０msごとですから，カーソルが反転するまでのインタバルは０．４秒，カーソル点滅の１周期は，０．８秒ということになります．

3）タイマ時刻レジスタ．＄Ｔ１のカウントアップを行います．なお，レジスタの各バイトの桁あふれ処理のために，ＴＭＤＡＴ（＄ＦＦ６Ｂ～＄ＦＦ６Ｅ）というデータテーブルが用意されており，それらの値は若いアドレスから順に，

５０（20ms×５０step＝1sec.　に相当）……１６進では＄３２
６０（1 sec.×６０　＝1 min.　）……１６進では＄３Ｃ
６０（1 min.×６０　＝1 hour　）……１６進では＄３Ｃ
２４（1 hour×２４　＝1 day　）……１６進では＄１８

となっております．この処理のはじめにＹレジスタをクリアしておきますが，これはＹレジスタが割込み要因の種類を１時的に蓄えておく機能を果たすためです．時刻レジスタ更新の結果，日付が変わった場合には，タイマ制御レジスタ．＄ＴＣの第３ビットをチェックし，０時割込みが要請されていればＹレジスタの第６ビットを立てます．

4）制御レジスタ第０ビットを見て予約時刻割込みがＯＮの場合には，．＄Ｔ１と．Ｔ１Ｉの比較を行って，等しい値ならばＹレジスタの第４ビットを立てます．

5）カウン　．＄Ｔ２のデクリメントを行い，その結果がゼロ（＄00000000）の場合には，再設定値レジスタ．＄Ｔ２Ｄから．＄Ｔ２への転送を行い（この動作をリストア処理ということがあります)，カウンタの値の再設定を行います．この後インタバル割込みがＯＮかどうかを確認し，ＯＮの場合には（．＄ＴＣ

の第１ビットが ″1″），Ｙレジスタの第５ビットを立てるとともに，制御レジスタ．$ＴＣのワンショット指定フラグ（第２ビット）が″1″のときには，インタバルをＯＦＦ（．$ＴＣの第１ビットをクリア）にします．

6）ＰＦキー番号レジスタ．ＰＦＩＲＱを読み込んで，０でない場合（ＰＦキー割込みあり）にはその値を下位４ビット，Ｙレジスタ上の第４ビットから第６ビットまでを，ＭＳＢを除く上位ビットとするステータス情報をＯＲ命令によって合成します．この際，ステータスレジスタＳＴＡＴＵＳ（共有ＲＡＭ上$Ｄ３８３）の第４～６ビットのうち，すでに″1″になっているビットについてはそのままになります．

また，ＰＦキー割込みが新たに生じていない場合には，ＳＴＡＴＵＳレジスタとＹレジスタとのＯＲをとることにより，新しいＳＴＡＴＵＳを合成します．つまりこの場合には，前に生じていたＰＦキー割込み番号が有効となるわけです．

7）最後に，ＳＴＡＴＵＳの第０～第７ビットのどれか１つでも立っている場合，すなわちメイン側から要請された割込み事象が１つでも発生している場合には，ＡＴＴＥＮＴ（Ｉ／Ｏポート：$Ｄ４０４番地）をリードして，メイン側に対してアテンション割込み要求を行います．サブシステムモニタプログラム内でのアテンション割込みの要求は，この場所でのみ行われます．なおアテンション割込みのハードウェア的な面については，第２章を参照して下さい．

以上のアルゴリズムを見てもらえばわかると思いますが，ユーザがＶＲＡＭアクセスフラグを立てて画面をアクセスする場合には，必ず前もってカーソル表示をストップ状態もしくは禁止状態にしておいてください．さもないと処理中にＮＭＩがかかった場合，ＣＤＳＰＣルーチンでアクセスフラグがクリアされてしまい，ＲＴＩ後の処理に不都合をきたす可能性があるからです．

なおソフトウェアの都合上，どうしてもＮＭＩ処理の開始を阻止したい場合には，タイマアクセスフラグ．ＴＭＡＣを頻繁に立て続けることです．ただしタイマレジスタの値は徐々に狂っていくことになります．

第4章

アプリケーションプログラム

サブシステムの使用にあたって――――

この章では，ユーザがサブシステムを使うときのための参考事項についてふれてみました．また，ユーザがサブシステムを使いやすくするためのアプリケーションや，それらを使ったサンプルをのせています．ユーザのプログラムを動作させるときは，この章および１章を参照してください．

PHASE III
SUBSYSTEM

4－1　サブシステムへのプログラム転送

サブシステム内でユーザのプログラムを実行するとき，プログラムを格納する場所として，次の領域があります．

1）共有ＲＡＭ　　　　　　　（＄Ｄ３８０～＄Ｄ３ＦＦ）
2）補助ワークエリア　　　　（＄Ｄ０Ａ０～＄Ｄ２ＢＦ）
3）コンソールバッファ　　　（＄Ｃ０００～＄ＣＦ９Ｆ）
4）ＶＲＡＭ　　　　　　　　（＄００００～＄ＢＦＦＦ）

これらの領域を使用するときには，制約が課されます．その理由は，これらの領域は普通の状態ではプログラムを格納すべき場所ではなく，データなどが置かれる場所であるからです．つまり，サブシステム内でユーザのプログラムが動くこと自体，かなり特殊な処理であるためです．この特殊な処理を行うため，各領域で動かす際の注意事項を簡単に説明します．

1）共有ＲＡＭ　（＄Ｄ３８０～＄Ｄ３ＦＦ）

共有ＲＡＭは１２８バイトありますが，プログラムを動かすためにはＴＥＳＴコマンドを用いなければならず，そのために最低１５バイト（ＴＥＳＴコマンドおよびＴＥＳＴサブコマンド）必要となります．従って，実際に使用できるのは＄Ｄ３８Ｆ（メイン側＄ＦＣ８Ｆ）から１１３バイトです．

他の領域を使用する場合も，共有ＲＡＭを通してプログラムを転送しなければなりません．このため，短い簡単なプログラムを動かす場合は，この領域を使用した方が，手間がかからないで済みますし，動作が終了した後は何の障害も起こりませんので，よいと思われます．一般的には共有ＲＡＭ上でプログラムを動かすことを推薦します．なお共有ＲＡＭを参照するサブルーチンは使用できません．

2）補助ワークエリア　（＄Ｄ０Ａ０～＄Ｄ２ＢＦ）

サブシステムで定常的に処理を行わせる場合は，共有ＲＡＭ上で充分ですが，メイン側とのデータをやり取りするときには，共有ＲＡＭ上の一部をＴＥＳＴコマンド，他の一部をプログラムエリア，他の一部をデータエリアとして使用することになります（ＴＥＳＴコマンドの一部をデータエリアとして使用することもできます）．

このとき，プログラムが大きくて，データエリアが少ししかとれない場合や，定常的な処理を行う行わないにかかわらず，プログラムが大きくて共有ＲＡＭ上

に収まりきらない場合は，この補助ワークエリアをプログラムの格納領域として用いるようにします．

補助ワークエリアは，全体で５４４（約0.5K）バイトあり，ＰＦキー定義テーブル（＄Ｄ２Ｃ０～＄Ｄ３５Ｆ：１６０バイト）も使用した場合は，全体で７０４バイトありますので，この領域で普通のプログラムは収まることと思います．ただし，ＰＦキー定義テーブルを使用するのは避けた方がよいでしょう．

補助ワークエリアを使うにあたっての制限は，サブシステムのモニタ内部ルーチンで，補助ワークエリアを使うサブルーチンをユーザのプログラムで用いてはならないというものです．また，ＰＦキー定義テーブルやキー入力バッファ（ＦＭ－７シリーズのみ）を用いる場合は，ＦＩＲＱをマスクしておく必要があります．なお，＄Ｄ０Ａ０～＄Ｄ０ＦＦ番地の領域は，ダイレクトアドレッシングモードでアクセスできますので，ユーザプログラムのワークエリアとして使用すると便利でしょう．

３）コンソールバッファ　（＄Ｃ０００～＄ＣＦ９Ｆ）

コンソールバッファは４０００（約４Ｋ）バイトありますので，プログラムが大きくなっても，この領域に収めることができます．この領域にプログラムを格納するときは，モニタ内のＰＵＴ，ＧＥＴコマンドの関連ルーチンはほとんど使用することができません．つまり，コンソールバッファをアクセスするサブルーチンはすべて用いることができません．

共有ＲＡＭか補助ワークエリアを使用する場合は，プログラムが終了した時点で何もする必要はないのですが，コンソールバッファを使用した場合は，終了した時点でＥＲＳＡＬＬ（＄Ｅ２５Ｅ）を呼び出して，バッファを初期設定しなければなりません（同時に画面も消えますから，画面が残るようにしたい場合は，ＥＲＳＢＵＦ（＄Ｅ３３１）とＡＳＳＦＬＤ（＄Ｅ２７Ｄ）を，必要な入力情報を与えて呼び出します）．

４）ＶＲＡＭ　（＄００００～＄ＢＦＦＦ）

プログラムの大きさが４Ｋバイト以上になる場合は，ＶＲＡＭ上にプログラムを置くようにします．このときは，モニタ内部のルーチンはほとんど用いない方がよいでしょう．プログラムは最大４８Ｋバイトまで格納できますが，１６Ｋバイト以上の長いプログラムを作ることは，まずありえないことですから，３画面のうち１画面を使うことになります．このときはメイン側のマルチページ制御レジスタ（＄ＦＤ３７）を制御して，プログラムの格納されている画面を表示さ

せないようにしておく必要があります．また，ＶＲＡＭアクセスフラグを始終セットしておかなければなりません．ＶＲＡＭは，ＣＲＴＣと共用のＲＡＭですから，ＦＭ－７７以外のサイクルスチール方式を採用していない機種では動作速度が低下します．

ＶＲＡＭ上にプログラムを置くのは避けた方がよいようです．もし，ＶＲＡＭを使うとしても，データエリアとして用いることにして，プログラム本体は別の領域に格納しておくようにした方がよいと思います．

プログラムをサブシステムにおく場合，可能性としては，以上の４つの領域がありますが，原則的には共有ＲＡＭ上で動作させるのが一番よい方法だと思います．実行速度が少々遅くても構わない場合は，プログラムをモジュール化して，分割して共有ＲＡＭ上で逐次実行させれば，大きなプログラムも共有ＲＡＭ上だけで動かすことは可能です．プログラムを常駐させるときも，補助ワークエリアかコンソールバッファにすべきで，ＶＲＡＭ上を使用するのはあまり良い方法とは思えません．

４－２　サブシステムでプログラムを動かすときの注意

サブシステム内でプログラムを動かす場合は，以下のようなことを注意して，プログラムを作成してください．

１）ＮＭＩ
２）ＶＲＡＭのアクセス
３）ワークエリア
４）メイン側とのインタフェース
５）リロケータブル（再配置可能）

各項目について簡単な説明を加えていきます．

１）ＮＭＩ

サブシステムでは，定常的にＮＭＩ（最優先割込み）が発生します．ＮＭＩサービスルーチンでは，カーソルの表示，タイマの更新，タイマやＰＦキーによるメイン側へのアテンション割込みなどの操作が行われています．操作を行うかどうかはワークエリアを見て判断しているので，ワークエリアを書き換えてしまうと，ＮＭＩでユーザにとっては不都合な処理を行うおそれがあるので気をつけて下さい．書き換えると，不都合を起こすワークエリアは次の２つです．

.CURSR　＄D01C　カーソルの表示をするためのフラグで，値が0かビット7がセットされているときは，カーソルを表示しませんが，それ以外の値をとるときは，カーソルを表示してしまいます．それだけではなく，カーソルを表示し終わるとVRAMアクセスフラグをオフにするので，ユーザのプログラムがVRAMをアクセスしているときに不都合が生じます．TESTコマンドでサブシステムに入った段階では表示させないようにしているので，書き換えない限り，問題は起こりません．

.＄TC　＄D00B　タイマ割込みを発生させるフラグで，0以外の値をとると，メイン側へアテンション割込みが発生します．ユーザプログラムを動かす前に，タイマ割込みが選択されていない場合は0となっています．

その他に，PFキー定義テーブルの各PFキーテーブルの先頭のバイトのビット7がセットされていると，PFキーを押せばNMIサービスルーチンによって，メイン側にアテンション割込みを発生させます．プログラムの格納場所としてPFキー定義テーブルを使うのは，このような理由から避けた方がよいでしょう．

2）VRAMのアクセス

VRAMをアクセスする前には，VRAMアクセスフラグをセットする（＄D409に対してread動作を行う）ことが必要です．BITA(＄D409)や TST（＄D409）などの操作を行って下さい．VRAMアクセスフラグをセットしておかないでVRAMをアクセスすると，CRTCとの競合が起こり，VRAM上に目的のデータを書き込む場合に誤動作します．特に表示されない領域（青の画面では＄3E80～＄3FFF，他の画面も同様）をアクセスするときに，VRAMアクセスフラグがセットされていないと，CRTCによって不要なデータがVRAM上に書き込まれる場合がありますので注意して下さい．

VRAMをアクセスしないときは，VRAMアクセスフラグをクリア（＄D409に対してwrite動作を行う）しておいた方が，CRTCによってCPUを止められることがないので良いでしょう．VRAMをアクセスしている間だけ，アクセスフラグをセットして下さい．

なお，VRAMアクセスフラグをセットした直後のVRAMへのアクセスは，ハードウェア的要因から避けて下さい．必ず，1命令以上実行してから，アクセスして下さい（すぐにアクセスしたいときは，NOP命令などを置いておくと良いでしょう）．

3）ワークエリア

ワークエリア（＄D000～＄D09F）には，定常的に用いられるものが多いので，遷移的に用いられているものは以外は書き換えることを避けた方がよいでしょう．参照する際は，別に問題はありません．もし，定常的に用いられているものを書き換えたい場合は，3章を読んで，そのワークエリアの内容を充分に知っておいて下さい．書き換えて使用したときは，プログラムの終了時点で元の値を戻しておいて下さい．

4）メイン側とのインタフェース

メイン側がサブシステムにHALTをかけた時に，コマンドを実行しないときは，＄D380（メイン側＄FC80）のビット7を立てておくようにします．ただし，ユーザのプログラム内での共有RAMを通じて情報の交換では，その必要はありません．メイン側が共有RAMをアクセスするときは，BUSYフラグ（メイン側＄FD05）を検索してBUSYでないとき，サブシステムにHALTをかけてから後にします．サブ側は，BUSY（＄D40A）をread／write動作を行って，メイン側に共有RAMをアクセスして良いか悪いかを知らせる必要があります．また，キャンセル割込みをかけるときは，メイン側の＄FD05のビット6をセットした後，もう一度ビット6をクリアします．このときサブ側ではIRQサービスルーチンによって，アボートフラグがセット（..ABRT：＄D000←＄FF）されますので，それをみてキャンセル割込みが起ったかどうか判断して下さい．なお，FM—8では，..IRQE（＄D000～＄D001）にジャンプベクトルを書き込むことによって，キャンセル割込みの処理先を指定することができます．

5）リロケータブル

サブシステムに置くプログラムは，サブシステム内で再配置しますから，システムに依存する部分（I／Oポートやモニタ内のサブルーチンを使うもの）を除いては，再配置可能なものでなくてはなりません．

４－３　サブシステムコールプログラム

仕　様　サブシステムにプログラムを送ったり，共有ＲＡＭで動かしたりするのは，面倒な手続きです．それらの手続きを代わりに行うためのプログラムです．ユーザはユーザプログラム中で必要なパラメータを設定し，ソフトウェア割込みで，このプログラムを呼びだすことにより，そのような手続きを一切行わないで，サブシステムを用いることができます．

使用法　プログラムは＄６０００～＄６２５１番地の領域にあります（リロケータブルです）．プログラムを呼び出すのは，ＳＷＩ（ソフトウェア割込み）を用いるのですが，Ｆ－ＢＡＳＩＣのＳＷＩジャンプテーブルを書き換えるため，使用する前にイニシァライザを出す必要があります．Ｆ－ＢＡＳＩＣから，

ＥＸＥＣ　＆Ｈ６０００ ⏎

とするか，ユーザのプログラムから，

ＪＳＲ　　＄６０００

とするかは，ユーザに任せることにします．

ユーザプログラムから，このプログラムを用いるときは，ＳＷＩを用います．このプログラムを呼び出す出には，次のレジスタを設定しなければなりません．

Ｘレジスタ　　パラメータの先頭番地
Ａレジスタ　　リクエスト番号（１～４）

また，呼び出した後は，次のレジスタとフラグが設定されます．

Ｃフラグ　　　エラーがあるときセットされる
Ａレジスタ　　エラーがあるときは，エラー番号（１～３）

このプログラムには，全部で４つのリクエストがあります．各リクエストと設定すべきパラメータを説明していきます．

１．ＣＡＬＬＥＲ　　リクエスト番号　１

サブシステムモニタ内，あるいはサブシステム内に移したユーザのプログラムを呼びだします．スイッチによって，呼びだす前のレジスタを設定したり，呼びだした後のレジスタの状態を戻したりすることができます．
パラメータは次のようになります．

相対値	内　　　　　　　　容
0，1	呼びだすサブルーチンのアドレス
2	サブルーチンを呼びだす際に，レジスタを設定するかどうかを決めるスイッチ　0…設定しない　1…設定する
3	サブルーチンを呼びだした後のレジスタの状態を戻すか戻さないかを決めるスイッチ　0…戻さない　1…戻す
4	呼びだすときのCCレジスタの値
5	呼びだすときのAレジスタの値
6	呼びだすときのBレジスタの値
7，8	呼びだすときのXレジスタの値
9，10	呼びだすときのYレジスタの値
11，12	呼び出すときのUレジスタの値

※相対値4～12の情報は，相対値2のスイッチを0にした場合は，設定する必要がありません．

レジスタの状態を戻すときは，次のような復帰情報が設定されます．

相対値	内　　　　　　　　容
4	呼び出した後のCCレジスタの値
5	呼び出した後のAレジスタの値
6	呼び出した後のBレジスタの値
7，8	呼び出した後のXレジスタの値
9，10	呼び出した後のYレジスタの値
11，12	呼び出した後のUレジスタの値

次の例は，サブシステムモニタ内のスクロールルーチンを呼びだすものです．

```
01000                                  NAM    SAMPLE  OF CALLER
01010                                  OPT    MEM
01020  5000                            ORG    $5000
01030                          *
01040  5000 30    8D 0004              LEAX   PARAM,PCR
01050  5004 86    01                   LDA    #1
01060  5006 3F                         SWI
01070  5007 39                         RTS
01080             5008         PARAM   EQU    *
01090  5008       E82E                 FDB    $E82E    'SCROLL
01100  500A       00                   FCB    $00      'NO REGISTER SET
01110  500B       00                   FCB    $00      'NO REGISTER RETURN
01120             5000                 END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=500B
PROGRAM ENTRY ADDR=5000
```

２．GO　　リクエスト番号　２

共有ＲＡＭ上にユーザのプログラムを持っていき，そのプログラムを動作させます．

パラメータは，次のようになります．

相対値	内　　　　容
０	プログラムの大きさ(１１３バイト以下)
１，２	プログラムの先頭アドレス(メインＲＡＭ内)

次の例は，“Ｆ”というキャラクタを表示させるものです．

```
                               NAM    SAMPLE  OF GO
                               OPT    MEM
      5000                     ORG    $5000
                         *
      5000 30   8D 0004        LEAX   PARAM,PCR
      5004 86   02             LDA    #2
      5006 3F                  SWI
      5007 39                  RTS
                5008     PARAM EQU    *
      5008      0A             FCB    EPROG-PROG 'PROGRAM SIZE
      5009      500B           FDB    PROG     'PROGRAM ADDRESS
                500B     PROG  EQU    *
      500B 86   46             LDA    #'F
      500D C6   07             LDB    #7
      500F 8E   2000           LDX    #$2000
      5012 7E   E51D           JMP    $E51D
                5015     EPROG EQU    *
                5000           END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=5014
PROGRAM ENTRY ADDR=5000
```

３．ＴＲＡＮＳ　　リクエスト番号　３

メイン側にあるデータを，サブ側に転送します．

パラメータは次のようになります．

相対値	内　　　　容
０，１	メイン側のデータの先頭アドレス
２，３	メイン側のデータの終了アドレス
４，５	サブ側の転送先の先頭アドレス

次の例は，メインのＲＯＭの内容を，ＶＲＡＭに書き込むものです．

```
                               NAM    SAMPLE  OF TRANS
                               OPT    MEM
      5000                     ORG    $5000
                         *
      5000 30   8D 0004        LEAX   PARAM,PCR
      5004 86   03             LDA    #3
      5006 3F                  SWI
```

```
        5007 39                   RTS
                   5008     PARAM EQU     *
        5008       8000           FDB     $8000    'SOURCE START  (MAIN)
        500A       FFFF           FDB     $FFFF    'SOURCE END    (MAIN)
        500C       0000           FDB     $0000    'DEST.  START  (SUB)
                   5000           END     $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=500D
PROGRAM ENTRY ADDR=5000
```

4．GET　　リクエスト番号　4

　サブ側にあるデータを，メイン側に転送します．
パラメータは，次のようになります．

相対値	内　　　　　　　容
0，1	サブ側のデータの先頭アドレス
2，3	サブ側のデータの終了アドレス
4，5	メイン側の転送先の先頭アドレス

次の例は，コンソールバッファの内容を，＄４０００～＄４Ｆ９Ｆ番地に読み込むものです．

```
                                  NAM     SAMPLE  OF GET
                                  OPT     MEM
        5000                      ORG     $5000
                            *
        5000 30    8D 0004        LEAX    PARAM,PCR
        5004 86    04             LDA     #4
        5006 3F                   SWI
        5007 39                   RTS
                   5008     PARAM EQU     *
        5008       C000           FDB     $C000    'SOURCE START  (SUB)
        500A       CF9F           FDB     $CF9F    'SOURCE END    (SUB)
        500C       4000           FDB     $4000    'DEST.  START  (MAIN)
                   5000           END     $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=500D
PROGRAM ENTRY ADDR=5000
```

エラー番号は，次のようになります．

番　号	内　　　　　　　容
1	リクエスト番号のエラー
2	プログラムサイズエラー(GOにおいて)
3	アドレスエラー(ＴＲＡＮＳ，ＧＥＴにおいて)

構　　造　各ルーチンごとに説明を加えていくことにします．

1） ＩＮＩＴ　　　Ｆ－ＢＡＳＩＣのＳＷＩジャンプテーブルを書き換えます．

2） ＨＡＮＤＬＥ　ＳＷＩサービスルーチンで，リクエストに応じて，各リクエストルーチンを呼びだします．

2） ＣＡＬＬＥＲ　ＣＡＬＬＥＲリクエストルーチンです．パラメータから，データを取り込み，共有ＲＡＭ上へ制御プログラムとパラメータを転送します．

3） ＣＡＬＬＳ　　ＣＡＬＬＥＲの，共有ＲＡＭ上で（サブ側で）動く制御ルーチンで，一緒に転送されたパラメータからレジスタを設定して，目的のルーチンを呼びだした後は，レジスタの内容を共有ＲＡＭ上に設定します．

4） ＧＯ　　　　　ＧＯリクエストルーチンです．

5） ＴＲＡＮＳＧ　Ｘレジスタで示される先頭アドレスから，Ｂレジスタの示すバイト数分だけプログラムを共有ＲＡＭへ転送し，そのプログラムをサブ側に実行させます．

6） ＴＲＡＮＳ　　ＴＲＡＮＳリクエストルーチンです．共有ＲＡＭ上にある制御プログラムと連絡をとりながら，メイン側にあるデータを共有ＲＡＭへ転送します．

7） ＴＲＡＮＳＳ　ＴＲＡＮＳのサブ側での制御プログラムです．共有ＲＡＭ上に転送されたデータをサブ側へ転送します．データは，６４バイトごとのブロックに区切られて転送されていきます．共有ＲＡＭの先頭のバイトをブロックのサイズに，次のバイトを転送のための連絡に使用しています．

8） ＭＯＶＥＲ　　ＴＲＡＮＳＳ用の，ブロック転送ルーチンです．

9） ＧＥＴ　　　　ＧＥＴリクエストルーチンです．共有ＲＡＭ上の制御プログラムと連絡をとりながら，共有ＲＡＭ上に転送されたデータをメイン側に転送します．

10） ＧＥＴＳ　　　ＧＥＴのサブ側での制御プログラムです．サブ側のデータを共有ＲＡＭ上へ転送します．データは，３２バイトごとのブロックに区切って転送してきます．共有ＲＡＭの先頭の次のバイトを，ブロックのサイズとしてメイン側との連絡に用いています．

11） ＭＯＶＥＧ　　ＧＥＴＳ用の，ブロック転送ルーチンです．

12） ＭＴＥＳＴ　　ＴＥＳＴコマンドを共有ＲＡＭ上に転送します．

13） ＭＴＥＳＴＳ　ＴＥＳＴサブコマンドを共有ＲＡＭ上へ転送します．

14) HALT　　サブシステムにHALTをかけます.

15) RHALT　　サブシステムにかけたHALTをリセットします.

16) COPY　　メイン側のブロック転送ルーチンです.

このプログラムの特徴は，サブ側とメイン側で同時に別々の処理を行っているということです(GET，TRANS)．サブとメインは，同時に共有RAMをアクセスするとき以外は，一方が他方を待っている必要はありません．また，転送の際は，ブロックごとに分けて，まとめて転送しているのも特徴です．

注意点　4－1節で述べた補助ワークエリアやコンソールバッファにユーザプログラムを転送して，そのプログラムを動かすようにするとき，このサブシステムコールプログラムを用いれば簡単に行うことができます．また，内部ルーチンを利用していないので，FM－8でも動作可能です．

```
*
**************************************************
*
*    Subsystem system caller
*
*    programed by t.minohara  may.1984
*
*
**************************************************
*
*            introduction
*
*  This application is made for the direct
* use of subsytem for user.
*  If user wants that he uses subroutine in
* subsystem , then he only calls this program
* with 'SWI' . He accsess it without making
* many subroutines each which uses maintenance
* (TEST) command .
*  This program has a variety of service .
* And user uses this same as using BIOS .
* Please make your program with this useful
* application !
*
*            method of use
*
*  method of call  'SWI'  software interrupt
*
*  before call , set next register
*
*  X  register .... databuffer of request
*  A  register .... request number
*
*  after call , set next register
*
*  CC register .... if error occured then C set
*  A  register .... error number
*
*                menu
*
*  1 . caller   call subsystem subroutine
*  2 . go       go program on communication area
*  3 . trans    transfer data or program to sub
*  4 . get      get data or program from sub
*
*                error
*
```

```
                                 *  1 . command error
                                 *  2 . program size error (in go)
                                 *  3 . address error
                                 *
                                 **************************************************
                                 *
                                 *  initializer
                                 *
                                 **************************************************
                                 *
                                         NAM     SUBCAL
                                         TTL     SUBSYSTEM CALLER
                                         OPT     MEM,NOGEN,NOW
6000                                     ORG     $6000
                                 *
              FC80               SRAM    EQU     $FC80
              FD05               SBSOUT  EQU     $FD05
                                 *
              D380               COMAR   EQU     $D380
              D409               VACC    EQU     $D409
              D40A               BUSY    EQU     $D40A
                                 *
                                 *  changer
                                 *
              01D7               SWIV    EQU     $01D7   * swi vector address
                                 *
              6000               INIT    EQU     *
6000 86       7E                         LDA     #$7E    * 'jmp' code
6002 B7       01D7                       STA     SWIV
6005 30       8D 0004                    LEAX    HANDLE,PCR * vector is handler entry
6009 BF       01D8                       STX     SWIV+1
600C 39                                  RTS
                                 *
                                 **************************************************
                                 *
                                 *  command handler
                                 *
                                 **************************************************
                                 *
              600D               HANDLE  EQU     *
                                 *
600D 4D                                  TSTA            * Is request 0 ?
600E 27       26    6036                 BEQ     ERROR   * Yes , then error .
6010 81       04                         CMPA    #4      * Is request great than 4 ?
6012 22       22    6036                 BHI     ERROR   * Yes , then error .
                                 *
6014 4A                                  DECA
6015 48                                  LSLA            * lbsr search
6016 31       8D 0020                    LEAY    VECTOR,PCR
601A 31       A6                         LEAY    A,Y     * get relative distance
601C EC       A4                         LDD     ,Y
601E 31       8D 0018                    LEAY    VECTOR,PCR
6022 31       AB                         LEAY    D,Y     * get absolute address
6024 AD       A4                         JSR     ,Y      * call each request routine
                                 *
6026 E6       E4                 FIN     LDB     ,S      * load CC register
6028 4D                                  TSTA
6029 27       04    602F                 BEQ     HANDL1
602B CA       01                         ORB     #1      * carry set
602D 20       02    6031                 BRA     HANDL2
602F C4       FE                 HANDL1  ANDB    #$FE    * carry reset
6031 E7       E4                 HANDL2  STB     ,S      * save CC register
                                 *
6033 A7       61                         STA     1,S
6035 3B                                  RTI
                                 *
                                 *  command error
                                 *
              6036               ERROR   EQU     *
6036 86       01                         LDA     #1      * command error code
6038 20       EC    6026                 BRA     FIN
                                 *
                                 *  vector
                                 *
              603A               VECTOR  EQU     *
603A          0008                       FDB     CALLER-VECTOR
603C          0091                       FDB     GO-VECTOR
603E          00B9                       FDB     TRANS-VECTOR
6040          014D                       FDB     GET-VECTOR
```

```
                         *
                         ***************************************************
                         *
                         *  caller
                         *
                         *       call subsystem subroutine
                         *
                         *       databuffer
                         *       +0 & +1    subruotine address
                         *
                         *    :: parameter in databuffer is registers which
                         *       set in time of calling the subroutine
                         *
                         *       databuffer
                         *       +2 set flag  0 : no register set
                         *                    1 : register set
                         *
                         *       if databuffer +2 is 1 then
                         *
                         *       databuffer +4      CC register
                         *                  +5      A  register
                         *                  +6      B  register
                         *                  +7 +8   X  register
                         *                  +9 +A   Y  register
                         *                  +B +C   U  register
                         *
                         *
                         *    :: if user expect then return
                         *       registers (after call) to databuffer
                         *
                         *       databuffer
                         *       +3 return flag  0 : not return
                         *                       1 : return value
                         *                           of register
                         *
                         *       if databuffer +3 is 1 then
                         *
                         *       databuffer +4      CC register
                         *                  +5      A  register
                         *                  +6      B  register
                         *                  +7 +8   X  register
                         *                  +9 +A   Y  register
                         *                  +B +C   U  register
                         *
          6042           CALLER EQU    *
                         *
6042 34   10                    PSHS   X
6044 A6   02                    LDA    2,X      * read set flag
6046 27   0B   6053             BEQ    CALL1
                         *
6048 30   04                    LEAX   4,X
604A 33   8D 0074               LEAU   REGIST,PCR
604E C6   09                    LDB    #9
6050 17   01F7 624A             LBSR   COPY     * transfer data
                         *
          6053           CALL1  EQU    *
6053 AE   E4                    LDX    ,S       * set subroutine address
6055 EC   84                    LDD    ,X
6057 ED   8D 0065               STD    ADDR,PCR
                         *
605B C6   4B                    LDB    #ENCAL-CALLS
605D 30   8D 001F               LEAX   CALLS,PCR
6061 17   006F 60D3             LBSR   TRANSG  * transfer program and go
                         *
6064 17   01CF 6236             LBSR   HALT     * wait for end of run
6067 AE   E4                    LDX    ,S
6069 A6   03                    LDA    3,X      * read return flag
606B 27   0C   6079             BEQ    CALL9
                         *
606D 8E   FCD1                  LDX    #SRAM+REGIST-CALLS+$0F
6070 EE   E4                    LDU    ,S
6072 33   44                    LEAU   4,U
6074 C6   09                    LDB    #9
6076 17   01D1 624A             LBSR   COPY     * return register
                         *
          6079           CALL9  EQU    *
6079 35   10                    PULS   X
607B 17   01C8 6246             LBSR   RHALT
607E 4F                         CLRA
```

```
607F 39                     RTS
                    *
                    *  program in shared ram
                    *
          6080      CALLS   EQU    *
6080 30   8D 003C           LEAX   ADDR,PCR
6084 AE   84                LDX    ,X
6086 34   10                PSHS   X
6088 30   8D 0036           LEAX   REGIST,PCR
608C A6   84                LDA    ,X
608E 1F   8A                TFR    A,CC
6090 34   01                PSHS   CC
6092 A6   01                LDA    1,X       * A  set
6094 E6   02                LDB    2,X       * B  set
6096 10AE 05                LDY    5,X       * Y  set
6099 EE   07                LDU    7,X       * U  set
609B AE   03                LDX    3,X       * X  set
609D 35   01                PULS   CC        * CC set
                    *
609F AD   F4                JSR    [,S]
                    *
60A1 34   12                PSHS   X,A
60A3 1F   A8                TFR    CC,A
60A5 30   8D 0019           LEAX   REGIST,PCR
60A9 A7   84                STA    ,X        * CC store
60AB E7   02                STB    2,X       * B  store
60AD 10AF 05                STY    5,X       * Y  store
60B0 EF   07                STU    7,X       * U  store
60B2 35   02                PULS   A
60B4 A7   01                STA    1,X       * A  store
60B6 EC   E1                LDD    ,S++
60B8 ED   03                STD    3,X       * X  store
                    *
60BA 35   10                PULS   X
60BC B5   D40A              BITA   BUSY
60BF 39                     RTS
                    *
                    *  buffer in shared ram
                    *
60C0      0002      ADDR    RMB    2
60C2      0009      REGIST  RMB    9
                    *
          60CB      ENCAL   EQU    *
                    *
                    *****************************************************
                    *
                    *  go
                    *
                    *       transfer program to shared ram
                    *       and go the program
                    *
                    *       databuffer
                    *
                    *       +0     : program size (<=113)
                    *       +1 +2  : program start address (in main)
                    *
          60CB      GO      EQU    *
                    *
60CB E6   80                LDB    ,X+       * read size
60CD C1   71                CMPB   #113      * size check
60CF 22   1B   60EC         BHI    GOO
60D1 AE   84                LDX    ,X        * read start address
                    *
          60D3      TRANSG  EQU    *
                    *
60D3 34   14                PSHS   X,B
60D5 17   015E 6236         LBSR   HALT      * wait for ready and halt
                    *
60D8 17   013C 6217         LBSR   MTEST     * transfer test command
60DB 30   8D 0010           LEAX   TESTS1,PCR
60DF 17   014C 622E         LBSR   MTESTS    * transfer test subcommand
                    *
60E2 35   14                PULS   B,X
60E4 17   0163 624A         LBSR   COPY      * transfer program
60E7 17   015C 6246         LBSR   RHALT     * reset halt
                    *
60EA 4F                     CLRA
60EB 39                     RTS
                    *
```

```
                60EC      GOO    EQU     *
                          *
6OEC 86         02               LDA     #2        * set size error
60EE 39                          RTS
                          *
                          *  test subcommand for go
                          *
                60EF      TESTS1 EQU     *
60EF            93               FCB     $93
60F0            D38F             FDB     COMAR+$0F
60F2            90               FCB     $90
                          *
                          ***************************************************
                          *
                          *  trans
                          *
                          *        transfer data from main to subsystem
                          *
                          *        databuffer
                          *
                          *        +0 +1  :  data start address in main
                          *        +2 +3  :  data end   address in main
                          *        +4 +5  :  data start address in sub
                          *
                60F3      TRANS  EQU     *
                          *
                          *  set parameter to stack and work
                          *  from databuffer
                          *
60F3 32         7C               LEAS    -4,S      * set parameter pool
60F5 EC         84               LDD     ,X
60F7 ED         E4               STD     ,S        * stack+0 = main.s.addr
60F9 EC         02               LDD     2,X
60FB 10A3       E4               CMPD    ,S
60FE 25         51    6151       BLO     TRANS0
6100 ED         62               STD     2,S       * stack+2 = main.e.addr
6102 EC         04               LDD     4,X
6104 ED         8D 007D          STD     SSADDR,PCR * (ssaddr)= sub.s.addr
                          *
6108 1A         50               ORCC    #$50      * FIRQ & IRQ mask
                          *
                          *  Set Parameter for TEST Command and
                          *        Program for Transfer
                          *        to Communication Area
                          *
610A C6         32               LDB     #ENTRAN-TRANSS
610C 30         8D 0045          LEAX    TRANSS,PCR
6110 17         FFC0 60D3        LBSR    TRANSG
                          *
                          *  Control Program in Main system
                          *       transfered data is blocked (64 bytes)
                          *
6113 AE         E4               LDX     ,S
                6115      TRANS1 EQU     *
6115 31         84               LEAY    ,X        * move data to shared ram
6117 C6         01               LDB     #1        * block size counter set
                          *
                6119      TRANS2 EQU     *
6119 10AC       62               CMPY    2,S       * compair to main.end.addr
611C 24         07    6125       BHS     TRANS3    * if pointer is great than
611E 5C                          INCB              * it then escape
611F 31         21               LEAY    1,Y       * inc pointer and counter
6121 C1         40               CMPB    #64       * if block size is small
6123 25         F4    6119       BLO     TRANS2    * than 64 then repeat
                          *
                6125      TRANS3 EQU     *
6125 CE         FCC0             LDU     #SRAM+64 * set block to shared ram
6128 17         010B 6236        LBSR    HALT
612B F7         FC80             STB     SRAM      * set block size
612E 7F         FC81             CLR     SRAM+1    * set data.ready flag
6131 17         0116 624A        LBSR    COPY      * move 1 block
6134 17         010F 6246        LBSR    RHALT
6137 30         84               LEAX    ,X        * Xreg=$0000 ?
6139 27         04    613F       BEQ     TRANS4    * Yes . then escape
613B AC         62               CMPX    2,S       * If we have not transfered
613D 23         D6    6115       BLS     TRANS1    * all data then repeat
                          *
                613F      TRANS4 EQU     *
613F 17         00F4 6236        LBSR    HALT
```

```
6142 7F   FC80             CLR    SRAM      * set end flag to $FC80
6145 7F   FC81             CLR    SRAM+1    * set data.ready flag
6148 17   00FB 6246        LBSR   RHALT
614B 4F                    CLRA
614C 32   64        TRANS5 LEAS   4,S       * erase stack pool
614E 1C   AF               ANDCC  #$AF      * FIRQ and IRQ enable
6150 39                    RTS
                    *
          6151      TRANS0 EQU    *
6151 86   03               LDA    #3        * data address error
6153 20   F7   614C        BRA    TRANS5
                    *
                    *  Control Program in Subsystem
                    *       accept blocked data and move it
                    *       to destination area
                    *
          6155      TRANSS EQU    *
                    *
6155 1A   50               ORCC   #$50      * FIRQ and IRQ mask
6157 B5   D409             BITA   VACC      * VRAM access set
615A EE   8D 0027          LDU    SSADDR,PCR
          615E      TRAN1  EQU    *
615E 8E   D3C0             LDX    #COMAR+64 * source data block addr
6161 C6   FF               LDB    #$FF      * reset data ready flag
6163 F7   D381             STB    COMAR+1
6166 F5   D40A             BITB   BUSY      * send ready signal (D40A)
                    *
6169 F6   D381      TRAN2  LDB    COMAR+1   * check data ready flag
616C 26   FB   6169        BNE    TRAN2
                    *
616E F7   D40A             STB    BUSY      * send busy flag
6171 F6   D380             LDB    COMAR     * load data block size
6174 27   04   617A        BEQ    TRAN3     * if empty then escape
6176 8D   05   617D        BSR    MOVER     * move data
6178 20   E4   615E        BRA    TRAN1     * repeat
                    *
          617A      TRAN3  EQU    *
617A 1C   AF               ANDCC  #$AF
617C 39                    RTS
                    *
                    *  mover  move data block
                    *
                    *          B ..block size
                    *          X ..source top address
                    *          U ..destination top address
                    *
          617D      MOVER  EQU    *
617D A6   80               LDA    ,X+
617F A7   C0               STA    ,U+
6181 5A                    DECB
6182 26   F9   617D        BNE    MOVER
6184 39                    RTS
                    *
6185      0002      SSADDR RMB    2         * Destinaion top address
                    *
          6187      ENTRAN EQU    *
                    *
                    ****************************************************
                    *
                    *  get
                    *
                    *          get data from subsystem to main
                    *
                    *          databuffer
                    *
                    *          +0 +1  :  data start address in subsystem
                    *          +2 +3  :  data end   address in subsystem
                    *          +4 +5  :  data start address in main
                    *
          6187      GET    EQU    *
                    *
6187 EC   04               LDD    4,X       * set main.s.addr to stack
6189 34   06               PSHS   D
618B EC   84               LDD    ,X        * set addr to work
618D ED   8D 0082          STD    SADDRS,PCR
6191 EC   02               LDD    2,X
6193 ED   8D 007E          STD    EADDRS,PCR
6197 10A3 8D 0077          CMPD   SADDRS,PCR
619C 25 . 28   61C6        BLO    GET0      * case of address error
```

```
                               *
619E 1A   50                           ORCC   #$50    * interrupt mask
                               *
61A0 30   8D 0028                      LEAX   GETS,PCR
61A4 C6   4B                           LDB    #ENGETS-GETS
61A6 17   FF2A 60D3                    LBSR   TRANSG
                               *
61A9 35   40                           PULS   U
          61AB                 GET1    EQU    *
61AB 17   0088 6236                    LBSR   HALT
61AE F6   FC81                         LDB    SRAM+1
61B1 27   0E   61C1                    BEQ    GET2
61B3 8E   FCE0                         LDX    #SRAM+96
61B6 17   0091 624A                    LBSR   COPY
61B9 7F   FC81                         CLR    SRAM+1
61BC 17   0087 6246                    LBSR   RHALT
61BF 20   EA   61AB                    BRA    GET1
                               *
          61C1                 GET2    EQU    *
61C1 17   0082 6246                    LBSR   RHALT
61C4 4F                                CLRA
61C5 39                                RTS
                               *
          61C6                 GET0    EQU    *
61C6 35   06                           PULS   D
61C8 86   03                           LDA    #3
61CA 20   F6   61C2                    BRA    GET2+1
                               *
                               *  Control Program in subsystem
                               *
          61CC                 GETS    EQU    *
                               *
61CC AE   8D 0043                      LDX    SADDRS,PCR
61D0 B5   D409                         BITA   VACC
61D3 1A   50                           ORCC   #$50
                               *
          61D5                 GETS1   EQU    *
61D5 31   84                           LEAY   ,X
61D7 C6   01                           LDB    #1
          61D9                 GETS2   EQU    *
61D9 10AC 8D 0037                      CMPY   EADDRS,PCR
61DE 24   07   61E7                    BHS    GETS3
61E0 5C                                INCB
61E1 31   21                           LEAY   1,Y
61E3 C1   20                           CMPB   #32
61E5 25   F2   61D9                    BLO    GETS2
                               *
          61E7                 GETS3   EQU    *
61E7 F7   D381                         STB    COMAR+1
61EA 8D   1C   6208                    BSR    MOVEG
61EC F5   D40A                         BITB   BUSY
                               *
          61EF                 GETS4   EQU    *
61EF F6   D381                         LDB    COMAR+1
61F2 26   FB   61EF                    BNE    GETS4
                               *
61F4 F7   D40A                         STB    BUSY
61F7 30   84                           LEAX   ,X
61F9 27   06   6201                    BEQ    GETS5
61FB AC   8D 0016                      CMPX   EADDRS,PCR
61FF 23   D4   61D5                    BLS    GETS1
                               *
          6201                 GETS5   EQU    *
6201 7F   D381                         CLR    COMAR+1
6204 F5   D40A                         BITB   BUSY
6207 39                                RTS
                               *
          6208                 MOVEG   EQU    *
6208 CE   D3E0                         LDU    #COMAR+96
620B A6   80                   MOVEG1  LDA    ,X+
620D A7   C0                           STA    ,U+
620F 5A                                DECB
6210 26   F9   620B                    BNE    MOVEG1
6212 39                                RTS
6213      0002                 SADDRS  RMB    2
6215      0002                 EADDRS  RMB    2
                               *
          6217                 ENGETS  EQU    *
                               *
                               **************************************************
                               *
```

```
                                 *  mtest    transfer test command to shared ram
                                 *           please halt subsystem before using
                                 *           this subroutine
                                 *
                    6217         MTEST  EQU     *
                                 *
        6217 30     8D 0008             LEAX    TESTD,PCR
        621B CE     FC80                LDU     #SRAM
        621E C6     0B                  LDB     #11
        6220 16     0027 624A           LBRA    COPY
                                 *
                                 *  test command data
                                 *
                    6223         TESTD  EQU     *
                                 *
        6223        0000                FDB     $0000
        6225        3F                  FCB     $3F      * test command
        6226        59                  FCC     'YAMAUCHI' * key word
                                 *
                                 ****************************************************
                                 *
                                 *  mtests   transfer test subcommand to shraed ram
                                 *           set halt before use
                                 *
                                 *           X reg ..top address of subcommand data
                                 *
                    622E         MTESTS EQU     *
                                 *
        622E CE     FC8B                LDU     #SRAM+11
        6231 C6     04                  LDB     #4
        6233 16     0014 624A           LBRA    COPY
                                 *
                                 ****************************************************
                                 *
                                 *  halt     wait ready and halt subsystem
                                 *
                    6236         HALT   EQU     *
                                 *
        6236 B6     FD05                LDA     SBSOUT  * wait for ready
        6239 2B     FB    6236          BMI     HALT
                                 *
        623B 86     80                  LDA     #$80    * set halt to subsystem
        623D B7     FD05                STA     SBSOUT
                                 *
        6240 B6     FD05         HALT2  LDA     SBSOUT  * wait until sub aknowledge
        6243 2A     FB    6240          BPL     HALT2   * halt signal
        6245 39                         RTS
                                 *
                                 ****************************************************
                                 *
                                 *  rhalt    reset halt
                                 *
                    6246         RHALT  EQU     *
                                 *
        6246 7F     FD05                CLR     SBSOUT
        6249 39                         RTS
                                 *
                                 ****************************************************
                                 *
                                 *  copy     copy data (<256)
                                 *
                                 *           X reg .. source top address
                                 *           U reg .. destination top address
                                 *           B reg .. source size
                                 *
                    624A         COPY   EQU     *
                                 *
        624A A6     80                  LDA     ,X+
        624C A7     C0                  STA     ,U+
        624E 5A                         DECB
        624F 26     F9    624A          BNE     COPY
        6251 39                         RTS
                                 *
                    6000                END     $6000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6000
PROGRAM END   ADDR=6251
PROGRAM ENTRY ADDR=6000
```

使用例　以下は，このプログラムを利用した例です．入力したアドレスから２５６バイト，サブシステムの内容をダンプします．GET（リクエスト番号４）を用いています．

```
10 '***********************************************
20 '*
30 '*  subsystem dumper
40 '*
50 '*
60 CLEAR,&H5FFF:WIDTH 80,25
70 EXEC&H6000
80 FOR I= 0 TO 12
90   READ A$:POKE&H5000+I,VAL("&H"+A$)
100 NEXT
110 '
120 START=-1
130 INPUT"dump start address :",START$
140 IF START$ ="" AND START<>-1 THEN 160 ELSE IF START$="" THEN 130
150 START=VAL("&H"+START$)
160 POKE&H5007,INT(START/256)
170 POKE&H5008,START-INT(START/256)*256
180 POKE&H5009,INT((START+255)/256)
190 POKE&H500A,START+255-INT((START+255)/256)*256
200 EXEC&H5000
210 '
220 PRINT"addr : +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F"
230 FOR I= 0 TO 15
240   PRINT RIGHT$("0000"+HEX$(START+I*16),4);" : ";
250   A$=""
260     FOR J = 0 TO 15
270       A=PEEK(&H5100+I*16+J):PRINT RIGHT$("00"+HEX$(A),2);" ";
280       IF A<&H20 OR A=&H7F THEN A$=A$+"." ELSE A$=A$+CHR$(A).
290     NEXT
300   PRINT " : "+A$
310 NEXT
320 '
330 INPUT "Do you continue ? (y/n)",A$
340 IF A$="N" OR A$="n" THEN END
350 START=START+256
360 GOTO 130
370 '*
380 DATA 8E,50,07    :'LDX #$5007  * parameter addr
390 DATA 86,04       :'LDA #4      * select GET
400 DATA 3F          :'SWI         * call subsystem caller
410 DATA 39          :'RTS
420 DATA 00,00       :'FDB $0000   * sub  start addr (dummy)
430 DATA 00,00       :'FDB $0000   * sub  end   addr (dummy)
440 DATA 51,00       :'FDB $5100   * main start addr
450 END
```

4－4　画面シフト

仕　様　このプログラムは，画面上の一部を横方向に１バイトまたは１ビットシフトするプログラムです．ただし完全な１ビットのシフト（任意のX座標，Y座標でのシフト）はできません．

使用法　このプログラムの最後に(このリストでは＄５１６５～＄５１６D)に次のように入力情報をセットして下さい．

＄５１６５～＄５１６６　X１座標
＄５１６７～＄５１６８　Y１座標
＄５１６９～＄５１６A　X２座標
＄５１６B～＄５１６C　Y２座標

（プログラムの位置によってアドレスは変化しますからプログラムを＄５０００以外に置くときは，オフセットをアドレスに加えて下さい．）

＄５１６D　シフトのパターン

では各々の入力情報を説明します．このプログラムは仕様でも述べたように完全にビット単位でのシフトはできません．シフトができる範囲を図で示します．図４・４・１を参照して下さい．X１，Y１，X２，Y２，はシフトしたい部分の対角線を取ります．しかし図のようにVRAM上のバイトの区切りのところでしかシフトできません．ですから図のように（X１，Y１）を選んだとすると，（X１＋１，Y１)，(X１＋２，Y１)，(X１＋３，Y１)，(X１＋４，Y１)，（X１＋５，Y１)，(X１＋６，Y１)，(X１＋７，Y１）と選んでも同様のシフトしかできません．X２についても同様です．つまり８ビットごとを切れ目にしてしかシフトできません．

X１，Y１，X２，Y２の値は，X１≦X２，Y１≦Y２のように選んで下さい．さらにこれらは画面の範囲を越えないように選んで下さい．

＄５１６Dについては，図４・４・２を参照して下さい．

図４・４・１

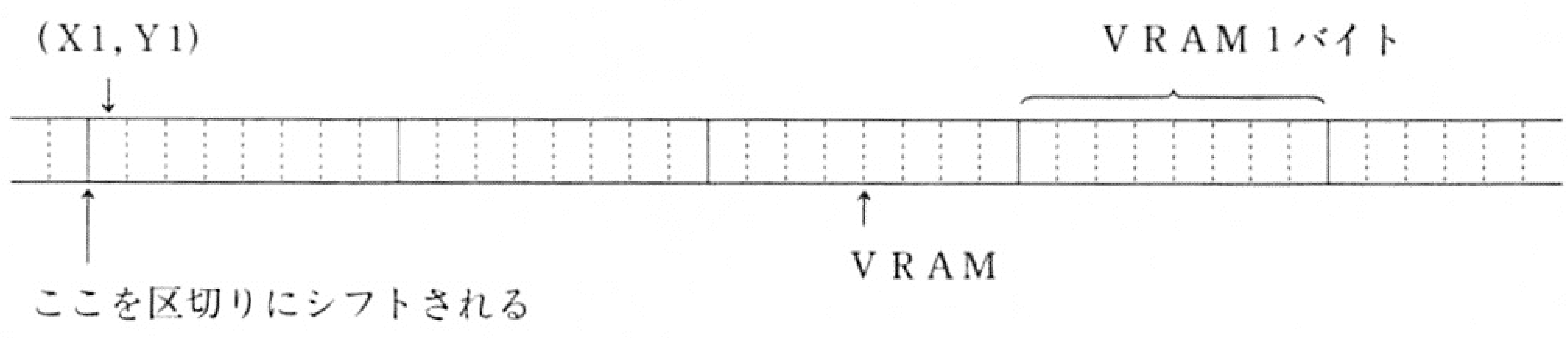


図4・4・2

＄５１６Ｄ	ビット1	ビット0
ビット単位の左シフト	0	0
バイト単位の左シフト	0	1
ビット単位の右シフト	1	0
バイト単位の右シフト	1	1

構　造　このプログラムは共有ＲＡＭ上で走らせるためなるべく短かく作りました．それでは説明していきます．まずcheck range(＄５０００～＄５０３７)でＸ１≦Ｘ２，Ｙ１≦Ｘ２，０≦Ｘ１，Ｘ２＜６４０，０≦Ｙ１，Ｙ２＜２００，でないかどうか調べます．そうでないときはプログラムをそこで終らせます．このようなルーチンはサブ側にもあるのですが共有ＲＡＭ上のプログラムを短かくするためにメイン側に置きました．次にＭＯＶＥで共有ＲＡＭ上にプログラムとデータを送りますが，テストコマンド，データ，メインルーチン，サブルーチンの順で送ります．サブルーチンは，シフトのパターンによって４つに分かれているのですが，この４つをすべて送ったのでは共有ＲＡＭがむだになります．そこであらかじめ＄５１６Ｄの値によって送るプログラムを決定するのです．ところがプログラムはそれぞれ長さが違いますから，このサブルーチンのあとにデータや他のプログラムを置くとアドレシングがうまくいきません．そのためにこの順に共有ＲＡＭ上へ転送するわけです．

次にサブ側のプログラムを説明します．ＰＲＧ２は，ＶＲＡＭのアドレスを求めるルーチンと青，赤，緑のＶＲＡＭを書き換えるプログラムです．

ＰＲＧ２は，１つのＶＲＡＭを書き換えるプログラムです．ここで注意してほしいのはＢＳＲ　ＰＲＧ３（＄５０Ｄ４～)とありますが，実際にＢＳＲするのは，ＰＲＧ３とは限らず，ＰＲＧ４，ＰＲＧ５，ＰＲＧ６ともなります．それはシフトのパターンによって共有ＲＡＭ上へのプログラム転送が行われているからです．

これは相対アドレシングの効果的な使用例です．

注意点　境界がＶＲＡＭ１バイトごとですから座標の計算に十分注意して下さい．なお改良点として任意のＸ１座標，Ｙ１座標でシフトするものが考えられますが，共有ＲＡＭ上に置くことは不可能ですから，他の場所に置くようなプログラムを作って下さい．

```
**************************************************
*
*   horizontal scroll
*
**************************************************
*
*            method of use
*
*        parameter
```

```
                          *       DATA+0:X1address
                          *       DATA+2:Y1address
                          *       DATA+4:X2address
                          *       DATA+6:Y2address
                          *       DATA+8:selection of shift
                          *          bit1=1:right shift
                          *          bit1=0:left shift
                          *          bit0=1:byte shift
                          *          bit0=0:bit shift
                          *
                          *       X1,Y1,X2,Y2 is two bytes
                          *
                          *
                          *
5000                              ORG     $5000
                                  OPT     MEM,NOGEN
                          *main
          FD05            SBSOUT  EQU     $FD05
          FC80            SHARED  EQU     $FC80
                          *sub
          D38F            SUBX1   EQU     $D38F
          D391            SUBY1   EQU     $D391
          D393            SUBX2   EQU     $D393
          D395            SUBY2   EQU     $D395
                          *sub subroutine
          E3DC            GADDR   EQU     $E3DC
                          *
          D409            VACC    EQU     $D409
                          *sub transient($D0A0)
          00A0            KEEPU   EQU     $A0
                          *
                          *  check range
                          *
5000 33   8D 0161                 LEAU    DATA,PCR
5004 AE   C1                      LDX     ,U++      *load X1
5006 10AE C1                      LDY     ,U++      *load Y1
5009 34   30                      PSHS    X,Y
500B 8C   0280                    CMPX    #$0280    *over range?
500E 24   25     5035             BHS     RTS       *yes
5010 108C 00C8                    CMPY    #$C8      *over range?
5014 24   1F     5035             BHS     RTS       *yes
5016 AE   C1                      LDX     ,U++      *load X2
5018 10AE C1                      LDY     ,U++      *load y2
501B 8C   0280                    CMPX    #$0280    *over range?
501E 24   15     5035             BHS     RTS       *yes
5020 108C 00C8                    CMPY    #$C8      *over range?
5024 24   0F     5035             BHS     RTS       *yes
5026 AC   E4                      CMPX    ,S        *X1>X2?
5028 25   0B     5035             BLO     RTS       *yes
502A 10AC 62                      CMPY    2,S       *Y1>Y2?
502D 25   06     5035             BLO     RTS       *yes
502F 1F   32                      TFR     U,Y
5031 32   64                      LEAS    4,S
5033 20   03     5038             BRA     MOVE
          5035            RTS     EQU     *
5035 32   64                      LEAS    4,S
5037 39                           RTS
                          ***************************************************
                          *
                          *  move programs and datas to shared ram
                          *
                          ***************************************************
          5038            MOVE    EQU     *
5038 8D   47     5081             BSR     HALT
                          *
                          * move test command
                          *
503A 30   8D 0052                 LEAX    PRG,PCR
503E CE   FC80                    LDU     #SHARED
5041 C6   0F                      LDB     #EPRG-PRG
5043 8D   34     5079             BSR     REPEAT
                          *
                          * move datas
                          *
5045 30   8D 011C                 LEAX    DATA,PCR
5049 C6   08                      LDB     #8
504B 8D   2C     5079             BSR     REPEAT
                          *
                          * move main program
```

```
                                  *
5051 30     8D 004E                      LEAX   PRG2,PCR
5051 C6     48                           LDB    #EPRG2-PRG2
5053 8D     24    5079                   BSR    REPEAT
                                  *
                                  * move sub program by selection
                                  *
5055 A6     A4                           LDA    ,Y
5057 44                                  LSRA
5058 24     0D    5067                   BCC    NOADD1
505A 30     8D 00E8                      LEAX   PRG6,PCR *load 1byte left shift
505E 44                                  LSRA
505F 24     04    5065                   BCC    NOADD3
5061 30     8D 00C3                      LEAX   PRG5,PCR *load 1byte right shift
            5065                  NOADD3 EQU    *
5065 20     0B    5072                   BRA    MOVE1
                                  *
            5067                  NOADD1 EQU    *
5067 30     8D 007C                      LEAX   PRG3,PCR *load 1bit left shift
506B 44                                  LSRA
506C 24     04    5072                   BCC    MOVE1
506E 30     8D 0096                      LEAX   PRG4,PCR *load 1bit right shift
                                  *
            5072                  MOVE1  EQU    *
5072 C6     21                           LDB    #33     *number of program's bytes
5074 8D     03    5079                   BSR    REPEAT
5076 8D     14    508C                   BSR    RHALT
5078 39                                  RTS
                                  *
                                  * move from Xreg to Ureg
                                  *
            5079                  REPEAT EQU    *
5079 A6     80                           LDA    ,X+
507B A7     C0                           STA    ,U+
507D 5A                                  DECB
507E 26     F9    5079                   BNE    REPEAT
5080 39                                  RTS
                                  *
            5081                  HALT   EQU    *
5081 B6     FD05                         LDA    SBSOUT  * wait for ready
5084 2B     FB    5081                   BMI    HALT
5086 86     80                           LDA    #$80    * halt subsystem
5088 B7     FD05                         STA    SBSOUT
508B 39                                  RTS
                                  *
            508C                  RHALT  EQU    *
508C 7F     FD05                         CLR    SBSOUT
508F 39                                  RTS
                                  *
                                  * program on shared ram
                                  *
            5090                  PRG    EQU    *
5090        0000                         FDB    $0000
5092        3F                           FCB    $3F
5093        59                           FCC    'YAMAUCHI'
509B        93                           FCB    $93
509C        D397                         FDB    $D397
509E        90                           FCB    $90
                                  *
                                  * main program
                                  *
            509F                  EPRG   EQU    *
            509F                  PRG2   EQU    *
509F 10BE   D391                         LDY    SUBY1   * load Y1
50A3 BE     D38F                         LDX    SUBX1   * load X1
50A6 BD     E3DC                         JSR    GADDR   * get Vram address
50A9 34     40                           PSHS   U
50AB BE     D393                         LDX    SUBX2   * load X2
50AE BD     E3DC                         JSR    GADDR   * get Vram address
50B1 35     10                           PULS   X
50B3 C6     03                           LDB    #3      * blue,red,green
50B5 F7     D38F                         STB    SUBX1
50B8 34     50                    JUMP3  PSHS   U,X
50BA 10BE   D391                         LDY    SUBY1
50BE 8D     12    50D2                   BSR    PRG1    * 1Vram shift
50C0 35     50                           PULS   U,X
50C2 30     89 4000                      LEAX   $4000,X * next Vram address
50C6 33     C9 4000                      LEAU   $4000,U * next Vram address
50CA 7A     D38F                         DEC    SUBX1
```

```
50CD 27   02   50D1        BEQ   EN      * all Vram shift ?
50CF 20   E7   50B8        BRA   JUMP3   * no
50D1 39             EN     RTS
                    *
                    * 1Vram shift
                    *
          50D2      PRG1   EQU   *
50D2 34   50        LOOP5  PSHS  U,X
50D4 8D   11   50E7        BSR   PRG3    * 1line shift
50D6 35   50               PULS  U,X
50D8 30   88 50            LEAX  80,X    * get next line add
50DB 33   C8 50            LEAU  80,U    * get next line add
50DE 31   21               LEAY  1,Y     * next line
50E0 10BC D395             CMPY  SUBY2   * Yreg>Y2
50E4 23   EC   50D2        BLS   LOOP5   * no
50E6 39                    RTS
          50E7      EPRG2  EQU   *
                    ************************************************
                    *
                    * 1line shift
                    *
                    ************************************************
                    *
                    *    parameter
                    *     Xreg :most left address
                    *     Ureg :most right address
                    *
                    *
                    *
                    * 1bit left shift
                    *
          50E7      PRG3   EQU   *
50E7 9F   A0               STX   KEEPU
50E9 B5   D409             BITA  VACC
50EC 12                    NOP
50ED A6   84               LDA   ,X
50EF 48                    ASLA
50F0 34   01               PSHS  CC
50F2 E6   C4        LOOP1  LDB   ,U
50F4 35   01               PULS  CC
50F6 59                    ROLB
50F7 34   01               PSHS  CC
50F9 E7   C4               STB   ,U
50FB 33   5F               LEAU  -1,U
50FD 1193 A0               CMPU  KEEPU
5100 24   F0   50F2        BHS   LOOP1
5102 35   01               PULS  CC
5104 B7   D409             STA   VACC
5107 39                    RTS
                    *
                    * 1bit right shift
                    *
          5108      PRG4   EQU   *
5108 DF   A0               STU   KEEPU
510A B5   D409             BITA  VACC
510D 12                    NOP
510E A6   C4               LDA   ,U
5110 47                    ASRA
5111 34   01               PSHS  CC
5113 E6   84        LOOP2  LDB   ,X
5115 35   01               PULS  CC
5117 56                    RORB
5118 34   01               PSHS  CC
511A E7   84               STB   ,X
511C 30   01               LEAX  1,X
511E 9C   A0               CMPX  KEEPU
5120 23   F1   5113        BLS   LOOP2
5122 35   01               PULS  CC
5124 B7   D409             STA   VACC
5127 39                    RTS
                    *
                    * 1byte right shift
                    *
          5128      PRG5   EQU   *
5128 DF   A0               STU   KEEPU
512A B5   D409             BITA  VACC
512D 12                    NOP
512E A6   C4               LDA   ,U
5130 E6   84               LDB   ,X
```

```
5132 A7    84              STA     ,X
5134 9C    A0        LOOP3 CMPX    KEEPU
5136 24    0A   5142       BHS     BYTE1
5138 30    01              LEAX    1,X
513A A6    84              LDA     ,X
513C E7    84              STB     ,X
513E 1F    89              TFR     A,B
5140 20    F2   5134       BRA     LOOP3
5142 B7    D409      BYTE1 STA     VACC
5145 39                    RTS
                     *
                     * 1byte left shift
                     *
           5146      PRG6  EQU     *
5146 9F    A0              STX     KEEPU
5148 B5    D409            BITA    VACC
514B 12                    NOP
514C A6    84              LDA     ,X
514E E6    C4              LDB     ,U
5150 A7    C4              STA     ,U
5152 1193  A0        LOOP4 CMPU    KEEPU
5155 23    0A   5161       BLS     BYTE2
5157 33    5F              LEAU    -1,U
5159 A6    C4              LDA     ,U
515B E7    C4              STB     ,U
515D 1F    89              TFR     A,B
515F 20    F1   5152       BRA     LOOP4
5161 B7    D409      BYTE2 STA     VACC
5164 39                    RTS
                     *
                     * parameter area
                     *
           5165      DATA  EQU     *
5165       0009            RMB     $9
                           END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=516D
PROGRAM ENTRY ADDR=****
```

使用例 以下はこのプログラムの使用例です．ワクの中の車が動きます．200行の値を0，1，2，3と変えて実行してみて下さい．

```
100 CLS
110 LINE@ (239,150)-(400,180),PSET,,B
120 CONNECT(340,176)-(338,170)-(330,166)-(320,166)-(315,170)-(301,171)-(300,176)-(340,176)
130 CIRCLE(310,176),6
140 CIRCLE(330,176),6
150 POKE &H5166,240
160 POKE &H5168,151
170 POKE &H5169,1
180 POKE &H516A,&H8F
190 POKE &H516C,179
200 POKE &H516D,0
210 FOR I=1 TO 1000
220 EXEC&H5000
230 NEXTI
240 END
```

4－5　画面スクロール

仕　様　Ｆ－ＢＡＳＩＣの拡張命令として，スクロールをさせるための命令を追加したものです．オフセットレジスタにオフセット値を書き込み画面をスクロールさせます．

使用法　プログラムは，＄６０００から＄６１７Ｄの領域にあります（リロケータブルです）．Ｆ－ＢＡＳＩＣの命令を拡張するため，イニシァライザを起動させます．

ＥＸＥＣ　＆Ｈ６０００⏎

拡張命令として，次の２つがあります．

ＳＣＲＯＬＬ［縦方向にスクロールするドット数］［，横方向にスクロールするドット数］
ＲＯＬＬ　　［オフセット増分値］

縦方向にスクロールするドット数の範囲は，－２００以上２００以下で，横方向は－６４０以上６４０以下です．横方向には，１２８ドット単位でしかスクロールできません．

［　］は省略可能なパラメータを示します．

オフセット増分値というのは，オフセットレジスタに足す増分値です．詳しくは，２－５節を参照して下さい．

構　造　各ルーチンごとに説明していきます．

1）INITIL　Ｆ－ＢＡＳＩＣを拡張するためのイニシァライザです．

2）MANAGE　Ｆ－ＢＡＳＩＣの拡張命令を制御するプログラムです．システム内部では，ＲＯＬＬの中間コードは＄Ｆ４，ＳＣＲＯＬＬの中間コードは＄Ｆ５に割りてられます．

3）SCROLL　ＳＣＲＯＬＬの制御プログラムです．文法をチェックした後，オフセット増分値を設定し，ＲＯＬＬＥＲを用いてスクロールを実行させます．

4）CALUC　縦方向，横方向のスクロールドット数からオフセット増分値を計算します．横方向のスクロールドット数を計算するときは，オフセット値のテーブル（ＯＦＴＢＬ１，ＯＦＴＢＬ２）を用いています．

5）ROLL　ＲＯＬＬの制御プログラムです．オフセット増分値を読み込

み，ＲＯＬＬＥＲを用いて実行させます．

6）ＲＯＬＬＥＲ　スクロール制御プログラムです．サブ側に制御プログラムを転送させ，それを動かします．

7）ＳＲＯＬＬ　ＲＯＬＬＥＲの，共有ＲＡＭ上に置かれるサブ側の制御プログラムです．２－５節で示した方式で，オフセットレジスタを書き換えます．

注意点　ＳＣＲＯＬＬで，縦方向にスクロールすると，２－５節で示した理由（ＶＲＡＭが中途半端であること）から，ずれが起こります．

```
                        *
                        *****************************************************
                        *
                        *        roll graphics
                        *
                        *         : roll & scroll command handler
                        *
                        *
                        *****************************************************
                        *
                                nam     roll
                                opt     mem,nogen
6000                            org     $6000
                        *
                        *  labels
                        *
            00D2        READ1   equ     $D2
            00D8        READ2   equ     $D8
            92A0        SYNTAX  equ     $92A0
            9A02        INTEG2  equ     $9A02
            9663        ILLEGL  equ     $9663
                        *
            0203        EXPNUM  equ     $0203
            0204        EXPTBL  equ     $0204
            0210        EXPMAN  equ     $0210
                        *
                        *  expand of F-BASIC statements
                        *
            6000        initil  equ     *
6000 86     08                  lda     #8
6002 B7     0203                sta     EXPNUM
6005 30     8D 000B             leax    table,pcr
6009 BF     0204                stx     EXPTBL
600C 30     8D 002C             leax    manage,pcr
6010 BF     0210                stx     EXPMAN
6013 39                         rts
                        *
                        *  statements table
                        *
            6014        table   equ     *
6014        43                  fcc     'CHAIﾎERASﾅLLISﾔLPRINﾔSOUNﾄPLAﾙ'
6032        52                  fcc     'ROLﾌSCROLﾌ'
                        *
                        *  manager
                        *
            603C        manage equ      *
603C 81     F4                  cmpa    #$F4
603E 1027   00D2 6114           lbeq    roll
6042 81     F5                  cmpa    #$F5
6044 27     03   6049           beq     scroll
6046 7E     92A0                jmp     SYNTAX
                        *
                        *****************************************************
                        *
                        *        scroll graphics
                        *
                        *        : scroll [up·down dot] [,left·right dot]
```

```
                        *
                        *
                        ****************************************************
                        *
           6049         scroll equu    *
                        *
                        *  read parameter from text
                        *
6049 4F                        clra           * default work area
604A 5F                        clrb
604B ED    8D 012C             std     updot,pcr
604F ED    8D 012A             std     ledot,pcr
                        *
6053 9D    D2                  jsr     READ1   * first parameter set ?
6055 27    21   6078           beq     scrol9 .* non parameter then end
6057 81    2C                  cmpa    #',
6059 27    0F   606A           beq     scrol1  * no . then skip
605B BD    9A02                jsr     INTEG2  * read first parameter
605E AF    8D 0119             stx     updot,pcr * set it to work area
6062 9D    D8                  jsr     READ2   * second parameter set ?
6064 27    0D   6073           beq     scrol2  * no . then skip
6066 81    2C                  cmpa    #',
6068 26    0F   6079           bne     scrol0
                        *
           606A         scrol1 equ     *
606A 9D    D2                  jsr     READ1   * pointer increment
606C BD    9A02                jsr     INTEG2  * read second parameter
606F AF    8D 010A             stx     ledot,pcr * set it to work area
                        *
                        *  call roller
                        *
           6073         scrol2 equ     *
6073 8D    07   607C           bsr     caluc   * cluculate parameter
6075 17    00B7 612F           lbsr    roller  * call roller
                        *
           6078         scrol9 equ     *
6078 39                        rts
                        *
                        *  on error
                        *
           6079         scrol0 equ     *
6079 7E    92A0                jmp     SYNTAX
                        *
                        ************************
                        *
                        *  caluculate parameter
                        *
                        ************************
                        *
           607C         caluc  equ     *
                        *
                        *  range check
                        *
607C AE    8D 00FB             ldx     updot,pcr * -200 <= updot <= 200 ?
6080 8C    00C8                cmpx    #200
6083 2E    76   60FB           bgt     caluc0  * no . then illegal
6085 8C    FF38                cmpx    #-200
6088 2D    71   60FB           blt     caluc0
                        *
608A AE    8D 00EF             ldx     ledot,pcr * -640 <= ledot <= 640 ?
608E 8C    0280                cmpx    #640
6091 2E    68   60FB           bgt     caluc0  * no . then illegal
6093 8C    FD80                cmpx    #-640
6096 2D    63   60FB           blt     caluc0
                        *
                        *  caluculate vertical offset
                        *
6098 6F    E2                  clr     ,-s     * make variable pool
609A 6F    E2                  clr     ,-s
609C AE    8D 00DB             ldx     updot,pcr * set up·down offset
60A0 CE    0000                ldu     #$0000
60A3 6D    8D 00D4             tst     updot,pcr
60A7 2B    0D   60B6           bmi     caluc2  * if updot < 0 then skip
                        *
60A9 30    1E           caluc1 leax    -2,x    * 2 dot up offset
60AB 8C    0000                cmpx    #0
60AE 2D    13   60C3           blt     caluc3
60B0 33    C9 00A0             leau    $a0,u   * add offset
60B4 20    F3   60A9           bra     caluc1
```

```
                          *
               60B6       caluc2 equ    *
60B6 30        02                leax   2,x      * 2 dot down offset
60B8 8C        0001              cmpx   #1
60BB 2C        06   60C3         bge    caluc3
60BD 33        C9 FF60           leau   -$a0,u   * sub offset
60C1 20        F3   60B6         bra    caluc2
                          *
60C3 EF        E4         caluc3 stu    ,s
                          *
                          *  caluculate horizontal offset
                          *
60C5 AE        8D 00B4           ldx    ledot,pcr
60C9 6D        8D 00B0           tst    ledot,pcr
60CD 2B        10   60DF         bmi    caluc5   * if ledot <0 then skip
                          *
60CF 33        8D 002B           leau   oftbl1,pcr
60D3 30        88 80      caluc4 leax   -128,x
60D6 8C        0000              cmpx   #0
60D9 2D        15   60F0         blt    caluc7
60DB 33        42                leau   2,u
60DD 20        F4   60D3         bra    caluc4
                          *
               60DF       caluc5 equ    *
60DF 33        8D 0027           leau   oftbl2,pcr
60E3 30        89 0080    caluc6 leax   128,x
60E7 8C        0000              cmpx   #0
60EA 2C        04   60F0         bge    caluc7
60EC 33        42                leau   2,u
60EE 20        F3   60E3         bra    caluc6
                          *
               60F0       caluc7 equ    *
60F0 EC        C4                ldd    ,u
60F2 35        10                puls   x
60F4 30        8B                leax   d,x
60F6 AF        8D 007F           stx    offset,pcr
60FA 39                          rts
60FB 7E        9663       caluc0 jmp    ILLEGL
                          *
                          *  offset table
                          *
               60FE       oftbl1 equ    *
60FE           0000              fdb    $00      0
6100           0060              fdb    $60      128
6102           0020              fdb    $20      256
6104           0080              fdb    $80      384
6106           0040              fdb    $40      512
6108           00A0              fdb    $a0      640
               610A       oftbl2 equ    *
610A           FFA0              fdb    -$60     -128
610C           FFE0              fdb    -$20     -256
610E           FF80              fdb    -$80     -384
6110           FFC0              fdb    -$40     -512
6112           FF60              fdb    -$a0     -640
                          *
                          ***************************************************
                          *
                          *        roll graphics
                          *
                          *        : roll [offset]
                          *
                          *
                          ***************************************************
                          *
                                 nam    roll
                          *
               6114       roll   equ    *
                          *
                          *  read parameter from text
                          *
6114 4F                          clra            * default work area
6115 5F                          clrb
6116 ED        8D 005F           std    offset,pcr
                          *
611A 9D        D2                jsr    READ1    * first parameter set ?
611C 27        0D   612B         beq    roll9    * non parameter then end
611E BD        9A02              jsr    INTEG2   * read first parameter
6121 AF        8D 0054           stx    offset,pcr * set it to work area
6125 9D        D8                jsr    READ2    * second parameter set ?
```

```
6127 26   03   612C        bne    roll0
                      *
                      *  call roller
                      *
6129 8D   04   612F        bsr    roller  * call roller
                      *
          612B        roll9  equ    *
612B 39                    rts
                      *
                      *  on error
                      *
          612C        roll0  equ    *
612C 7E   92A0             jmp    SYNTAX
                      *
                      ************************
                      *
                      *  roller
                      *
                      ************************
                      *
          612F        roller equ    *
                      *
                      *  labels
                      *
          FC80        SHARED equ    $FC80
          FD05        HALT   equ    $FD05
                      *
          001C        CURSR  equ    $1C        (subsystem)
          001F        VOFST  equ    $1F        (subsystem)
                      *
          D380        COMAR  equ    $D380      (subsystem)
          D409        VACC   equ    $D409      (subsystem)
          D40E        SOFFST equ    $D40E      (subsystem)

612F 1A   40               orcc   #$40     * IRQ mask
                      *
                      *  halt subsystem
                      *
6131 B6   FD05        rolle1 lda    HALT
6134 2B   FB   6131        bmi    rolle1   * wait ready flag
6136 86   80               lda    #$80     * halt to subsystem
6138 B7   FD05             sta    HALT
                      *
613B B6   FD05        rolle2 lda    Halt
613E 2A   FB   613B        bpl    rolle2   * wait for ready
                      *
                      *  transfer test command and program
                      *
6140 30   8D 0012          leax   data,pcr * pointer set
6144 C6   25               ldb    #endata-data * counter set
6146 CE   FC80             ldu    #SHARED
                      *
6149 A6   80          rolle3 lda    ,x+
614B A7   C0               sta    ,u+
614D 5A                    decb
614E 26   F9   6149        bne    rolle3
                      *
                      *  subsystem go
                      *
6150 7F   FD05             clr    HALT     * reset HALT
6153 1C   BF               andcc  #$BF     * IRQ enable
6155 39                    rts
                      *
                      *  test command
                      *
          6156        data   equ    *
6156      0000             fdb    $0000
6158      3F               fcb    $3F      * test command
6159      59               fcc    'YAMAUCHI' * key word (dummy)
                      *
                      *  test subcommand
                      *
6161      93               fcb    $93
6162      D38F             fdb    sroll-data+COMAR
6164      90               fcb    $90
                      *
                      *  roller on subsystem
                      *
          6165        sroll  equ    *
```

```
6165 0F    1C                    clr    CURSR    * cursor enable
6167 EC    8D 000E               ldd    offset,pcr * read offset value
616B D3    1F                    addd   VOFST    * add to vram offse
616D B5    D409                  bita   VACC     * vram access flag on
6170 DD    1F                    std    VOFST    * set it to work area
6172 FD    D40E                  std    SOFFST   * set it to i/o
6175 B7    D409                  sta    VACC     * vram access flag off
6178 39                          rts
                          *
6179       0002           offset rmb    2
                          *
           617B           endata equ    *
                          *
                          *  work area
                          *
617B       0002           updot  rmb    2
617D       0002           ledot  rmb    2
                          *
           6800                  end    $6800
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=6000
PROGRAM END   ADDR=617E
PROGRAM ENTRY ADDR=6800
```

使用例　ＳＣＲＯＬＬを用いた簡単なサンプルです．

```
1000 CLS
1010 CIRCLE(320,100),100,2,,,,F
1020 FOR I= 0 TO 50
1030   CIRCLE(320,100),160+I,5,.1,,.63+(I/1300)
1040   CIRCLE(320,100),160+I,5,.1,,.87-(I/1300),1
1050 NEXT
1060 S=4*((RND>.5)+.5)
1070 FOR I= 0 TO 100
1080   SCROLL S
1090 NEXT
1100 GOTO 1060
```

4－6　画面拡張

仕　様　青の画面の指定された枠の中の内容を，２倍に拡大して赤の画面に転送します．

使用法　プログラムは，＄５０００～＄５１０６番地の領域にあります（リロケータブル）です．ユーザのプログラムから，サブルーチンとして呼びだします．

　　ＪＳＲ　＄５０００

このとき，Ｘレジスタにパラメータの先頭アドレスを入れておきます．
パラメータは，次のようになります．

相対値	内　　容
0	水平方向の拡大スイッチ　０で拡大せず，０以外で２倍に拡大
1	垂直方向の拡大スイッチ　０で拡大せず，０以外で２倍に拡大

2，3	始点のX座標　X1
4，5	始点のY座標　Y1
6，7	終点のX座標　X2
8，9	終点のY座標　Y2

※X座標は，8の倍数でなければなりません．

拡大された内容は，赤の画面の同一始点の枠に転送されます（図4・6・1）．2倍に拡大したとき，終点が画面からはみ出しそうなときは，動作を行いません．次のような条件になるように配慮して下さい．

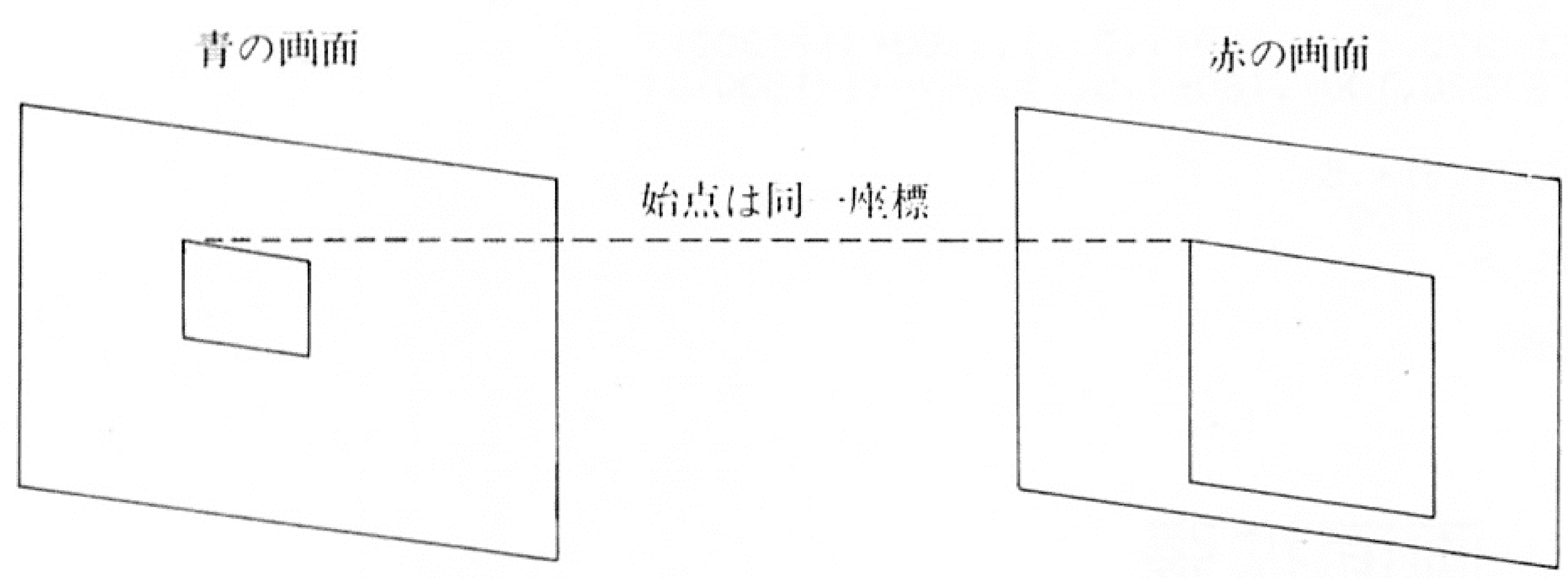
水平方向に拡大するとき

X2＋（X2－X1）×2　≦　639

垂直方向に拡大するとき

Y2＋（Y2－Y1）×2　≦　199

図4・6・1　拡大の様子



構　造　各ルーチンごとに説明します．

1）MAIN　Xレジスタがその先頭番地を示すパラメータを，ワークエリアに読み込み，各座標が範囲内にあるかどうかチェックします．範囲内にあるときは，制御プログラムをワークエリアごと，共有RAM上へ転送して，動かせます．

2）CHECK　与えられた座標が，範囲内にあるかどうかチェックします．

3）COPY　ブロック転送ルーチン

4）PROG　共有RAM上で動く制御プログラムです．垂直方向の拡大を主に制御します．

5）LEXP　水平方向の拡大を制御するサブルーチンです．

モニタ内部のルーチンとして，GADDR（$E3DC），BITEXT（$E50B）を用いています．

注 意 点　青の画面から赤の画面への定位置での拡大でしたが，$50A9～

＄５０ＡＡ番地と＄５０Ｄ２～＄５０Ｄ３番地を書き換えることにより，任意の画面から任意の画面の任意の位置へ拡大することができます．

```
                              ***************************************************
                              *
                              *   expand  pattern  of  blue  VRAM
                              *                        to  red    VRAM
                              *
                              *   scale is 1:2 set next parameters
                              *
                              *   Xreg+0 horizn  (horizontal expand switch)
                              *         =0 .. 1:1  /  <>0 .. 1:2
                              *
                              *   Xreg+1 verti   (vertical expand switch)
                              *         =0 .. 1:1  /  <>0 .. 1:2
                              *
                              *   Xreg+2 x1   start position x-co.ordinate (2bytes
                              *   Xreg+4 y1   start position y-co.ordinate (2bytes
                              *   Xreg+5 x2   end   position x-co.ordinate (2bytes
                              *   Xreg+7 y2   end   position y-co.ordinate (2bytes
                              *
                              *   !  x1,x2 is multiple of 8
                              *
                                      NAM     EXPAND
                                      OPT     MEM,NOGEN
5000                                  ORG     $5000
                              *
               FC80           SHARED EQU      $FC80
               FD05           SBSOUT EQU      $FD05
                              *
               D409           VACC    EQU     $D409
               E3DC           GADDR   EQU     $E3DC
               E50B           BITEXT EQU      $E50B
                              *
               5000           MAIN    EQU     *
                              *
                              *  read parameter & set them to work
                              *
5000 33        8D 00F9                LEAU    HORIZN,PCR
5004 C6        0A                     LDB     #10
5006 17        0080 5089              LBSR    COPY
                              *
                              *  check range
                              *
5009 AE        8D 00F2                LDX     X1,PCR  * check range of source
500D AC        8D 00F2                CMPX    X2,PCR
5011 22        74    5087             BHI     ERROR
5013 10AE 8D 00E9                     LDY     Y1,PCR
5018 10AC 8D 00E8                     CMPY    Y2,PCR
501D 22        68    5087             BHI     ERROR
501F 8D        58    5079             BSR     CHECK
5021 AE        8D 00DE                LDX     X2,PCR
5025 10AE 8D 00DB                     LDY     Y2,PCR
502A 8D        4D    5079             BSR     CHECK
                              *
502C CC        0000                   LDD     #0
502F 6D        8D 00CA                TST     HORIZN,PCR * check range of
5033 27        0E    5043             BEQ     MAIN1    * destination
5035 EC        8D 00CA                LDD     X2,PCR
5039 A3        8D 00C2                SUBD    X1,PCR
503D 58                               ASLB             * d=d*2
503E 49                               ROLA
503F E3        8D 00BC                ADDD    X1,PCR
5043 1F        01             MAIN1   TFR     D,X      * x=x1+(x2-x1)*2
5045 CC        0000                   LDD     #0
5048 6D        8D 00B2                TST     VERTI,PCR
504C 27        0E    505C             BEQ     MAIN2
504E EC        8D 00B3                LDD     Y2,PCR
5052 A3        8D 00AB                SUBD    Y1,PCR
5056 58                               ASLB
5057 49                               ROLA
5058 E3        8D 00A5                ADDD    Y1,PCR
505C 1F        02             MAIN2   TFR     D,Y      * y=y1+(y2-y1)*2
505E 8D        19    5079             BSR     CHECK
                              *
                              *  transport program to shared ram
```

```
                         *
5060 B6   FD05           MAIN3   LDA     SBSOUT  * halt
5063 2B   FB    5060             BMI     MAIN3
5065 86   80                     LDA     #$80
5067 B7   FD05                   STA     SBSOUT
                         *
506A 30   8D 0023                LEAX    PROG,PCR
506E CE   FC80                   LDU     #SHARED
5071 C6   76                     LDB     #EPROG-PROG
5073 8D   14    5089             BSR     COPY
                         *
5075 7F   FD05                   CLR     SBSOUT  * run the program
5078 39                          RTS
                         *
                         *  check range
                         *
          5079           CHECK   EQU     *
5079 8C   027F                   CMPX    #639
507C 22   07    5085             BHI     ERROR0
507E 108C 00C7                   CMPY    #199
5082 22   01    5085             BHI     ERROR0
5084 39                          RTS
          5085           ERROR0  EQU     *
5085 32   62                     LEAS    2,S       * abandon return addr
          5087           ERROR   EQU     *
5087 43                          COMA
5088 39                          RTS
                         *
                         *  copy
                         *
          5089           COPY    EQU     *
5089 A6   80                     LDA     ,X+
508B A7   C0                     STA     ,U+
508D 5A                          DECB
508E 26   F9    5089             BNE     COPY
5090 39                          RTS
                         *
                         *  test command in shared ram
                         *
          5091           PROG    EQU     *
5091      0000                   FDB     $0000
5093      3F                     FCB     $3F
5094      59                     FCC     #YAMAUCHI#
509C      93                     FCB     $93
509D      D38F                   FDB     $D38F
509F      90                     FCB     $90
                         *
                         *  program
                         *
50A0 B5   D409                   BITA    VACC
50A3 10AE 8D 0059                LDY     Y1,PCR
50A8 8E   4000                   LDX     #$4000  * offset load
50AB 8D   18    50C5     EXPAN1  BSR     LEXP
50AD 6D   8D 004D                TST     VERTI,PCR
50B1 27   05    50B8             BEQ     EXPAN2  * if not expand then skip
50B3 30   88 50                  LEAX    80,X    * +80 to offset
50B6 8D   0D    50C5             BSR     LEXP
50B8 31   21             EXPAN2  LEAY    1,Y
50BA 10AC 8D 0046                CMPY    Y2,PCR
50BF 23   EA    50AB             BLS     EXPAN1
50C1 B7   D409                   STA     VACC
50C4 39                          RTS
                         *
                         *  line expand
                         *
50C5 34   30             LEXP    PSHS    X,Y
50C7 AE   8D 0034                LDX     X1,PCR
50CB BD   E3DC                   JSR     GADDR   * get vram addr
50CE EC   E4                     LDD     ,S      * read offset
50D0 33   C9 0000                LEAU    >$0000,U
50D4 31   CB                     LEAY    D,U     * Y is destination addr
50D6 A6   C4             LEXP1   LDA     ,U
50D8 6D   8D 0021                TST     HORIZN,PCR
50DC 27   0E    50EC             BEQ     LEXP2
50DE BD   E50B                   JSR     BITEXT  * case of expand
50E1 E7   21                     STB     1,Y
50E3 BD   E50B                   JSR     BITEXT
50E6 E7   A4                     STB     ,Y
50E8 31   22                     LEAY    2,Y
```

```
50EA 20    04   50F0           BRA    LEXP3
50EC A7    A4           LEXP2  STA    ,Y          * case of normal
50EE 31    21                  LEAY   1,Y
50F0 30    08           LEXP3  LEAX   8,X
50F2 33    41                  LEAU   1,U
50F4 AC    8D  000B            CMPX   X2,PCR
50F8 23    DC   50D6           BLS    LEXP1
50FA 35    30                  PULS   X,Y
50FC 39                        RTS
                        *
                        *  works
                        *
50FD       0001         HORIZN RMB    1           * horizontal expand switch
50FE       0001         VERTI  RMB    1           * vertical   expand switch
50FF       0002         X1     RMB    2           * expand start x-co.ord
5101       0002         Y1     RMB    2           * expand start y-co.ord
5103       0002         X2     RMB    2           * expand end   x-co.ord
5105       0002         Y2     RMB    2           * expand end   y-co.ord

           5107         EPROG  EQU    *
           5000                END    $5000
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=5106
PROGRAM ENTRY ADDR=5000
```

使用例 簡単なサンプルです．機械語部分をＲＡＭに設定してから動作します．

```
1000 '*
1010 '* demo of expand
1020 '*
1030 FOR I= &H4000 TO &H400F
1040  READ A$:POKE I,VAL(˝&H˝+A$)
1050 NEXT
1060 '
1070 CLS:SCREEN,1
1080 COLOR=(2,5):COLOR=(1,5)
1090 CIRCLE (15,7),15,1,,,,F
1100 '
1110 FOR I=1 TO 9
1120   IF (I MOD 2)=0 THEN 1170
1130     POKE&H50A9,&H40:POKE&H50D2,&H00
1140     EXEC&H4000
1150     SCREEN,2:GOTO 1200
1160 '
1170     POKE&H50A9,&HC0:POKE&H50D2,&H40
1180     EXEC&H4000
1190     SCREEN,1
1200 NEXT
1210 CLS:COLOR=(2,2):COLOR=(1,1):SCREEN7,7
1220 END
1230 '*
1240 '* machine language data
1250 '*
1260 DATA 8e,40,06,7e,50,00
1270 DATA 01,01                :'horizn,verti
1280 DATA 00,00,00,00          :'x1,y1
1290 DATA 01,3e,00,62          :'x2,y2
1300 '*
1310 '*  if you want to select vram
1320 '*  then set it
1330 '*
1340 '               $50A9    $50D2
1350 'blue ->red      $40      $00
1360 'blue ->green    $80      $00
1370 'red  ->blue     $C0      $40
1380 'red  ->green    $40      $40
1390 'green->blue     $80      $80
1400 'green->red      $C0      $80
```

第5章

FM-7/NEW7/77とFM-8との比較

FM-8ユーザへの指針――――

本書はFM－7シリーズについての解析書ですが，FM－8のユーザの皆さんにも御活用頂けるように，FM－7とFM－8の比較をまとめたのがこの章です．この比較を御覧になっていただくと，FM－7のために記述・製作されたものを容易にFM－8に移殖できるようになります．また新たに発表されたNEW－7とFM－77では，サブシステム内の相違はありませんが，メインROM内において，少々の改良が施されていますので，それについてもふれることにしました．FM－NEW7／77も含めて，FM－7／8シリーズのユーザの皆さんの御健闘を期待致します．

5－1　FM－7とFM－8のサブシステムの比較

5－1－1　FM－7とFM－8のサブシステムの比較

FM－7とFM－8では，サブシステムのアルゴリズムはほとんど同じです．ですからアドレスを変えるだけで，FM－7用のプログラムをFM－8でも動作させることができます．しかし，若干の相違がありますので以下に説明して行くことにします．

主な相違は，次のようになります．

1）キー入力バッファの有無
2）IRQの処理
3）補助ワークエリア
4）アルゴリズムの改良

各々について，簡単な説明を加えて行きます．

1）キー入力バッファの有無

FM－8では，キー入力バッファは有りません．そこで，KEYに関するサブルーチンが変わっています．また，KEYを使って制御を行うルーチンも異っています．ワークエリアもこの理由によって若干の違いが出ています．

この理由によって，次のようなルーチンが変わっています．

RESTAR	($E000)	PUT010	($E5EB)
ESCSEQ	($E7FB)	GETDEL	($EBE8)
GCURS	($F67D)	INKEY	($FD5C)
KEYIN0	($FD76)	INCKBX	($FD95)
KBFERA	($FD9E)	KEYIRQ	($FDAC)
KSTBUF	($FDE6)	KEYR2	(FM－8)

※　アドレスは，FM－7のものです．

2）IRQの処理

FM－8では，IRQに対する処理ルーチンへのベクトルをワークエリア(..IRQE)に持っています．ですから，このワークエリアを書きかえることによって，IRQの処理をユーザのプログラムで行うことができます．FM－8のサブシステムモニタ内では，コマンド待ちルーチンやGETコマンドルーチン，Graphic Cursor コマンドルーチンなど，Breakキー（キャンセル割込み）を扱う処理系では，そのコマンドルーチン内のアボートルーチンへジャンプさせるようにベクトルを書きか

えています．FM－7では，アボートフラグ（..ABRT）が設けてあり，IRQが起こったときはそれがセットされますので，各ルーチンはアボートフラグを参照して処理を行っています．

この理由によって，次のようなルーチンが変っています．

IRQAB	($E06E)	CMDWAB	(FM－8)
CMDWAI	($E133)	PUTDW	($E7DE)
GET000	($E930)	GETABT	(FM－8)
GC0002	($F6F6)		

3）補助ワークエリア

FM－7では，FM－8で補助ワークエリア内で持っていた情報（ウィンドウの範囲）を，普通のワークエリア内で持つことにしたため，そのワークエリアを設定するためのルーチンを毎回呼ばなくてすむようになり，その分プログラムが短くなっています．また，PAINTとGet Blockの補助ワークエリアを重ね合わせることによって，一つのサブルーチン（CLEXT）で処理できるようにしています．この理由によって，次のルーチンが異っています．

CHGWIN	($E1F2)	ERASE0	($E298)
PUT000	($E5E4)	GETC	($EC46)
GBLK1	($F286)	CLEXT	($F2ED)
PAINT	($FA8A)		

4）アルゴリズムの改良

FM－7では，機能が増えた分だけ，サブルーチンを最適化しなければなりませんでした．例えば，FM－8では数ステップかけて実行していた動作を，FM－7ではサブルーチンを呼びだすことによって，1ステップで行ったりしています．また，PAINTとGet Blockにおいて，FM－8では別々に行っていたカラーエクステンドテーブルの作成を一つのサブルーチンで行っています（補助ワークエリアの項を参照してください）．また，Graphic Cursorでは，グラフィックカーソルの移動ルーチンを呼ぶときにジャンプテーブルが短くなるように改良しています．この理由によって，次のルーチンが異っています．

CMDCTL	($E034)	PUTCHR	($E678)
PUTINC	($E7BA)	GETINS	($E9D2)
CURSET	($EDE8)	GBLK1	($F286)
CLEXT	($F2ED)	GC000B	($F756)
PAINT	($FA8A)		

5）その他

ＦＭ－８では，ＰＦキー表示モードのときでも画面のスクロールをオフセットレジスタを用いる高速スクロールで行っていますが，ＦＭ－７では，ＰＦキー表示モードのときは，ＶＲＡＭの内容を転送する方法で行っています．これは，後者の方法で行っても，ＦＭ－７では十分速いし，スクロールする度にＰＦキーを表示しなおす（ＦＭ－８）と画面が散らつくからです．また，ＴＥＳＴコマンドのキーワード検索をＦＭ－７では行っていませんので，その分ＴＥＳＴコマンドルーチンが短くなっています．これらの理由で次のルーチンが異っています．

ＳＣＲＯＬＬ　（＄Ｅ８２Ｅ）　　　ＴＥＳＴ　（＄ＦＦ６Ｆ）

これらの相違は，ＦＭ－８のサブシステムモニタVersion４．０との比較です．ＦＭ－８発売開始前に供給されたVersion３．３（一般のユーザは持っていないはずです）では，このほかにもたくさんの相違があります．例えば，Version３．３では，ＰＡＩＮＴの境界色をドット単位で検索していたり，ＬＩＮＥでＸ軸，Ｙ軸に平行なときの場合わけをしていなかったりします．このため，グラフィックは，Version３．３ではだいぶ遅くなっています．

５－１－２　サブシステムのワークエリア，アドレス対照表

ＦＭ－７とＦＭ－８のサブシステムのワークエリアの対照表とモニタ内のサブルーチンのアドレスの対照表を以下に掲げます．これは，メモリマップとしても参照して下さい．アドレス対照表の相違の項目では，アルゴリズム的に異るものに対して，相違があると判定しています．相違のないものはワークエリアとサブルーチンのアドレスを変更すれば，たいていはそのまま使えるものです．相違があるものでも，少々変更すれば動作するようになります（そのまま使えるものもあります）．

サブシステム・ワークエリアのアドレス対応表

FM－7…V6.0JR　　FM－8…V4.0

ラベル名	FM－7	FM－8	バイト数	内容
..ABRT	D000	―――	1	アボートフラグ
..IRQE	―――	D000	2	IRQベクトル
..CONT	D001	D002	1	CONTフラグ保存用
KLOCK	D002	―――	1	キー入力禁止(キーロック)フラグ
KBFFLG	D003	―――	1	キーバッファフラグ
KCOUNT	D004	―――	1	キー入力数
KHEAD	D005	―――	2	キー入力バッファ読取り用ポインタ
KTAIL	D007	―――	2	キー入力バッファ書込み用ポインタ
KEYDAT FLAG	―――	D003	1	キー入力フラグ（＝0キー入力なし，≠0キー入力あり，ビット7＝1PEキー）
KEYDAT CODE	―――	D004	1	キー入力コード(PFキーのときは，PFキー文字数)
PFIX	―――	D005	2	PFキーの内容を取り出すためのポインタ
.PFIRQ	D009	D007	1	PFキー割込み番号
.TMAC	D00A	D008	1	タイマアクセスフラグ
.$TC	D00B	D009	1	タイマ制御レジスタ
.$T1	D00C	D00A	4	タイマ時刻レジスタ
.$T1I	D010	D00E	4	割込み時刻レジスタ
.$T2	D014	D012	4	タイマカウンタ
.$T2D	D018	D016	4	タイマカウンタ再設定値レジスタ
.CURSR	D01C	D01A	1	カーソル表示フラグ
.CURTM	D01D	D01B	1	カーソルタイマ
.CURST	D01E	D01C	1	カーソル状態フラグ
.VOFST	D01F	D01D	2	VRAMオフセット値
.CSCTL	D021	D01F	1	コンソール制御フラグ
PUTCFG	D022	D020	1	PUT継続フラグ
CSLCDS	D023	D021	1	単色表示フラグ
.TABST	D024	D022	10	TAB停止位置設定テーブル
..ACL	D02E	D02C	1	現在のアトリビュートコード

ラベル名	FM－7	FM－8	バイト数	内　　　容
..BCL	D02F	D02D	1	バックカラーコード
..CLEX	D030	D02E	1	キャラクタフォント出力時のベースイメージ
..CWD	D031	D02F	1	キャラクタ幅フラグ
..CHT	D032	D030	1	キャラクタアンダーマージンフラグ
..INS	D033	D031	1	インサートモードフラグ
.EKEY	D034	D032	1	終了キー番号
..LNCD	D035	D033	2	1行の桁数(2バイトとして用いるとき)
..LNC	D036	D034	1	1行の桁数
..DNL	D037	D035	1	画面の行数
..FS	D038	D036	1	PFキー表示フラグ
..VCU	D039	D037	2	VRAM上の桁単位
..VLU	D03B	D039	2	VRAM上の行単位
..DMAP	D03D	D03B	8	画面情報(各画面の開始行と終了行)
.BPY	D045	D043	1	バッファポインタのY座標
.BPX	D046	D044	1	バッファポインタのX座標
.BPAD	D047	D045	2	バッファポインタのキャラクタバッファアドレス
.BAAD	D049	D047	2	バッファポインタのアトリビュートバッファアドレス
.BVAD	D04B	D049	2	バッファポインタのVRAM上のアドレス
.MDRG	D04D	D04B	3	ウィンドウ画面情報(開始行，終了行，画面番号)
GTTOPC	D050	――	2	ウィンドウ画面の先頭のキャラクタバッファアドレス
GTBTMC	D052	――	2	ウィンドウ画面の終了のキャラクタバッファアドレス
GTTOPA	D054	――	2	ウィンドウ画面の先頭のアトリビュートバッファアドレス
GTBTMA	D056	――	2	ウィンドウ画面の終了のアトリビュートバッファアドレス
.CY	D058	D04E	1	カーソルのY座標
.CX	D059	D04F	1	カーソルのX座標
.CBAD	D05A	D050	2	カーソルのキャラクタバッファアドレス
.CAAD	D05C	D052	2	カーソルのアトリビュートバッファアドレス
.CVAD	D05E	D054	2	カーソルのVRAM上のアドレス
.GCL	D060	D056	1	グラフィックカラーコード
.GF	D061	D057	1	グラフィックファンクションコード

ラベル名	FM－7	FM－8	バイト数	内　　　　　容
.GY	D062	D058	2	ドット表示用Y座標
.GX	D064	D05A	2	ドット表示用X座標
.GAD	D066	D05C	2	グラフィックアドレス(未使用)
.GBMK	D068	D05E	1	グラフィックブビットマーク(未使用)
.BOXX1	D069	D05F	2	対角線座標　X1
.BOXY1	D06B	D061	2	対角線座標　Y1
.BOXX2	D06D	D063	2	対角線座標　X2
.BOXY2	D06F	D065	2	対角線座標　Y2
INDLP1	D069	D05F	1	ループのカウンタ(INDICTルーチン)
INDLP2	D06A	D060	1	未使用
INDLP3	D06B	D061	1	ループのカウンタ(PFDSPルーチン)
INDPF	D06C	D062	1	PFキー表示文字数
INDBSV	D06D	D063	1	PFキー表示時の，元のバックカラー保存用
.LX1	D071	D067	2	ライン　X1座標
.LY1	D073	D069	2	ライン　Y1座標
.LX2	D075	D06B	2	ライン　X2座標
.LY2	D077	D06D	2	ライン　Y2座標
.CBTOP	D079	D06F	2	消去開始バッファアドレス(未使用)
.CBBTM	D07B	D071	2	消去終了バッファアドレス＋1(未使用)
.CVTOP	D07D	D073	2	消去VRAM先頭アドレス（現在は，消去する行の開始行と終了行）
.CVBTM	D07F	D075	2	消去VRAM終了アドレス(未使用)
.ERSIX	D081	D077	2	未使用
BLUSID	D079	D06F	3	VRAMデータ転送用フック(青)
REDSID	D07C	D072	3	VRAMデータ転送用フック(赤)
GRESID	D07F	D075	3	VRAMデータ転送用フック(緑)
EXFONT	D083	D079	18	キャラクタフォント拡張データ保存用
BLUDSP	D083	D079	3	ドット表示用フック(青)
REDDSP	D086	D07C	3	ドット表示用フック(赤)
GREDSP	D089	D07F	3	ドット表示用フック(緑)
BLULIN	D08C	D082	3	ライン出力用フック(青)

ラベル名	FM－7	FM－8	バイト数	内　容
REDLIN	D08F	D085	3	ライン出力用フック(赤)
GRELIN	D092	D088	3	ライン出力用フック(緑)
HKFLG	D095	D08B	2	キャラクタフォント出力時の文字色と背景色の保存用
HKBLUE	D097	D08D	3	キャラクタフォント出力用フック(青)
HKRED	D09A	D090	3	キャラクタフォント出力用フック(赤)
HKGREE	D09D	D093	3	キャラクタフォント出力用フック(緑)
.WORK	D0A0	D096	544/554	補助ワークエリア
.PFTBL	D2C0	D2C0	160	PFキー定義テーブル
KEYBUF	D360	―――	32	キー入力バッファ

サブシステム・モニタ・サブルーチンのアドレス対照表

ＦＭ－７…Ｖ６.０ＪＲ　　ＦＭ－８…Ｖ４.０

ラベル名	ＦＭ－７	ＦＭ－８	※相違	機　能
ＲＥＳＴＡＲ	Ｅ０００	Ｅ０００	×	サブシステムイニシァライズ(リセット後の初期化)
ＣＭＤＣＴＬ	Ｅ０３４	Ｅ０２７	×	コマンドハンドラ
ＣＭＳＥＮＤ	Ｅ０６Ｃ	Ｅ０６２	－	コマンドハンドラの終了番地
ＩＲＱＡＢ	Ｅ０６Ｅ	Ｅ０６４	×	キャンセル割込み(ＩＲＱ)サービス
ＣＭＤＰＮ	Ｅ０７６	Ｅ０６Ｆ	－	コマンドハンドラ用ジャンプテーブル
ＩＮＰＡＲＭ	Ｅ０Ｃ７	Ｅ０Ｃ０	○	ＩＮＩＴ用デフォルトパラメータ
ＥＲＲ６０～	Ｅ０ＣＦ	Ｅ０Ｃ８	○	エラーエントリ群
ＥＲＲ７０	Ｅ０ＥＤ	Ｅ０Ｅ６	○	無効コマンドエラーエントリ
ＥＲＲＯＲ	Ｅ０ＥＦ	Ｅ０ＥＢ	○	エラー処理
ＣＯＮＴＯ	Ｅ０Ｆ５	Ｅ０ＥＥ	○	ＣＯＮＴコマンド待ち(メイン側へ継続データ出力)ルーチン
ＣＯＮＴＯ２	Ｅ１０５	Ｅ０ＦＥ	○	エラー，コマンドコードレジスタ，ＣＯＮＴフラグのクリア
ＣＮＴＡＢＴ	Ｅ１１５	Ｅ１０Ｅ	○	ＣＯＮＴコマンドのキャンセル（他のコマンドを受け付けたときの処理)
ＣＯＮＴＩ	Ｅ１２１	Ｅ１１Ａ	○	ＣＯＮＴコマンド待ち（メイン側から継続データ入力）ルーチン
ＣＯＭＩＮＴ	Ｅ１２Ａ	Ｅ１２３	○	ＣＯＮＴフラグの保存，エラー，コマンドコードレジスタ，ＣＯＮＴフラグのクリア
ＣＭＤＷＡＢ	―――	Ｅ１２Ｃ	×	コマンド待ちルーチンのアボート(Abort)ルーチン
ＣＭＤＷＡＩ	Ｅ１３３	Ｅ１２Ｆ	×	コマンド待ちルーチン（コマンドコードをＡレジスタに)
ＩＮＩＴ	Ｅ１４Ｅ	Ｅ１５Ｂ	○	ＩＮＩＴ(画面の初期化)コマンドルーチン
ＤＥＦＡＬＴ	Ｅ１ＣＢ	Ｅ１Ｄ８	○	デフォルト値による画面の初期化
ＨＯＭＥ	Ｅ１ＤＡ	Ｅ１Ｅ７	×	ウィンドウ画面の決定，初期化
ＣＨＧＷＩＮ	Ｅ１Ｆ２	―――	×	ウィンドウ画面の初期化
ＢＦＰＴＲＬ	Ｅ１ＦＡ	Ｅ１ＦＦ	○	Ｘ座標０として，バッファポインタのアドレスを求める
ＢＵＦＰＴＲ	Ｅ１ＦＣ	Ｅ２０１	○	バッファポインタのバッファアドレス，ＶＲＡＭ上のアドレスを求める
ＢＢＡＤＲ	Ｅ２００	Ｅ２０５	○	バッファポインタのバッファアドレスを求め，ワークエリアに設定する
ＢＡＤＲＬ	Ｅ２０８	Ｅ２０Ｄ	○	ＡレジスタがＹ座標，Ｘ座標０のバッファアドレスを求める
ＢＡＤＲＮ	Ｅ２０Ａ	Ｅ２０Ｆ	○	バッファポインタのバッファアドレスを求める
ＢＡＤＲ	Ｅ２０Ｃ	Ｅ２１１	○	キャラクタＸＹ座標からバッファアドレスを求める

※相違とは，ＦＭ－７とＦＭ－８間での相違を示し，○で相違無し，×で相違有りを示します．

ラベル名	ＦＭ－７	ＦＭ－８	※相違	機　　　　　能
ＢＶＡＤＲ	Ｅ２１Ｃ	Ｅ２２１	○	バッファポインタのＶＲＡＭ上のアドレスを求め，ワークエリアに設定する
ＢＶＡＤＬ	Ｅ２２１	Ｅ２２６	○	Ａレジスタが Ｙ座標，Ｘ座標０のＶＲＡＭ上のアドレスを求める
ＢＶＡＤＮ	Ｅ２２３	Ｅ２２８	○	バッファポインタのＶＲＡＭ上のアドレスを求める
ＢＶＡＤ	Ｅ２２５	Ｅ２２Ａ	○	キャラクタＸＹ座標からＶＲＡＭ上のアレスを求める
ＮＥＲＡＳＥ	Ｅ２４１	Ｅ２１６	○	ＥＲＡＳＥ２（画面の消去，初期化）コマンドルーチン
ＥＲＡＳＥ	Ｅ２４Ｂ	Ｅ２５０	○	ＥＲＡＳＥ（画面の消去，初期化）コマンドルーチン
ＥＲＳＡＬＬ	Ｅ２５Ｅ	Ｅ２６３	○	全画面を消去する
ＡＳＳＦＬＤ	Ｅ２７Ｄ	Ｅ２８２	○	各画面の先頭にフィールドの先頭を設定する
ＥＲＡＳＥ０	Ｅ２９８	Ｅ２９Ｄ	×	指定画面を消去，アトリビュートコード設定
ＥＲＡＳＥ１	Ｅ２Ａ０	Ｅ２Ａ５	×	指定画面を消去する
ＡＳＭＦＬＤ	Ｅ２Ｃ５	Ｅ２Ｃ９	○	指定画面の先頭にフィールドの先頭を設定する
ＥＲＳＶＲＭ	Ｅ２Ｄ３	Ｅ２Ｄ７	○	画面（ＶＲＡＭ）上の指定行を消去する
ＥＲＳＳＯＦ	Ｅ３２５	Ｅ３２９	○	画面上の指定行を消去（コンソールバッファおよびＶＲＡＭ）
ＥＲＳＳＵＢ	Ｅ３２８	Ｅ３２Ｃ	○	画面上の指定行を消去（消去最中に画面をオフにしない）
ＥＲＳＢＵＦ	Ｅ３３１	Ｅ３３５	○	指定行のコンソールバッファを消去，初期化
ＥＲＳＡＲＡ	Ｅ３４８	Ｅ３４Ｃ	○	指定領域を指定データで埋める
ＥＲＳＶ	Ｅ３６８	Ｅ３６Ｃ	○	ＶＲＡＭの指定領域を消去する
ＧＡＸＣ	Ｅ３８３	Ｅ３８７	○	グラフィックＸＹ座標の範囲チェック（ワークエリアに対して）
ＧＡＸＣＲ	Ｅ３８８	Ｅ３８Ｃ	○	グラフィックＸＹ座標の範囲チェック
ＢＡＳ	Ｅ３９２	Ｅ３９６	○	ビットイメージデータ
ＧＰＧＥＴＲ	Ｅ３９Ａ	Ｅ３９Ｅ	○	画面上のあるドットのカラーを調べる
ＣＦＡＤＲ	Ｅ３Ｂ７	Ｅ３ＢＢ	○	キャラクタフォントのデータのアドレスを求める
ＣＢＡＤＲ	Ｅ３Ｃ３	Ｅ３Ｃ７	○	カーソルのバッファアドレス，ＶＲＡＭ上のアドレスを求める
ＣＡＸＣＲ	Ｅ３Ｄ５	Ｅ３Ｄ９	○	キャラクタＸＹ座標の範囲のチェック
ＧＡＤＤＲ	Ｅ３ＤＣ	Ｅ３Ｅ０	○	グラフィックＸＹ座標からＶＲＡＭ上のアドレスを求める
ＧＸＯＦＦ	Ｅ３Ｆ２	Ｅ３Ｆ６	○	Ｘ座標からＶＲＡＭ上のオフセット位置（８の整数除算），ビット位置（８の剰余）を求める
ＧＸＯＦＦＲ	Ｅ３Ｆ４	Ｅ３Ｆ８	○	８による整数除算，剰余を行う
ＬＳＲＤ３	Ｅ３ＦＢ	Ｅ３ＦＦ	○	８による整数除算を行う
ＢＯＸＩＮＶ	Ｅ４０２	Ｅ４０６	○	対角線座標を共有ＲＡＭからワークエリアに読み込む
ＢＯＸ	Ｅ４１Ａ	Ｅ４１Ｅ	○	対角線座標の範囲のチェック（ワークエリアに対して）

※○でアルゴリズムの相違無し，Ｘで相違有り

ラベル名	FM－7	FM－8	※相違	機　　　能
BOXR	E41D	E421	○	対角線座標の範囲のチェック
GAXCHX	E442	E446	○	グラフィックXY座標の範囲のチェック（範囲外のときは，エラールーチンへ）
MOVE	E44A	E44E	○	ブロック転送ルーチン
INITLN	E452	E456	○	ファンクションコード，カラーコードからLINEのフックを設定する
GPDSPI	E469	E46D	○	GPDSPのフックの設定（ワークエリアに基づいて）
GPDSIR	E46B	E46F	○	GPDSPのフックの設定
FENTRY	E46E	E472	○	ファンクションのフックの設定(PBLK1より)
LENTRY	E471	E475	○	LINEのフックの設定(INITLNより)
GPDSP	E48F	E493	○	指定された座標（ワークエリアに基づいて）に点をうつ
GPDSPC	E494	E498	○	指定された座標に点をうつ
GPDSPR	E497	E49B	○	指定されたVRAM上のアドレス，ビットに点をうつ
GPDFNC	E4AD	E4B1	－	点の表示サブルーチン群へのフック用のジャンプベクトルデータ
GPORA～	E4C1	E4C5	○	点の表示サブルーチン群
CDSINT	E4E6	E4EA	○	CDSPのフックの設定
BITEXT	E50B	E50F	○	Aレジスタの下位4ビットを拡張してBレジスタに格納
CDSPC	E518	E51C	○	カーソルの位置にキャラクタを表示する
CDSPB	E51B	E51F	○	バッファポインタの位置にキャラクタを表示する
CDSP	E51D	E521	○	指定位置へキャラクタを表示する
CDSRVL～	E58F	E593	○	キャラクタのフォント表示サブルーチン群
CDJMP	E5D0	E5D4	－	フォント表示サブルーチン群へのフック用のジャンプベクトルデータ
GET	E5E0	E5E4	×	GET(文字列入出力)コマンドルーチン
PUT	E5E1	E5E5	×	PUT(文字列出力)コマンドルーチン
PUT000	E5E4	E5E8	×	PUTメインルーチン
PUT010	E6EB	E5F2	×	PUTメインループ
PUTSPC	E676	E661	×	バッファポインタの示す位置に空白を出力する
PUTCHR	E678	E663	×	バッファポインタの示す位置に1文字出力する
PUTOEL	E6A0	E684	○	PUTのEL(Erase Line)オーダルーチン
PUTOBL	E6A8	E68C	○	PUT，GET（Bell）オーダルーチン（GETBEL）
PUTOCL	E6AC	E690	○	PUTのEA(Erase All)オーダルーチン
PUTOCR	E6B4	E698	○	PUTのCR(Carriage Return)オーダルーチン

※○でアルゴリズムの相違無し，×で相違有り

ラベル名	FM−7	FM−8	※相違	機　　　　能
PUTOLF	E6BE	E6A2	○	PUTのLF(Line Feed)オーダルーチン
PUTOHM	E6C6	E6AA	○	PUTのHOMEオーダルーチン
PUTOSF	E6CE	E6B2	○	PUTのSF(Start Field)オーダルーチン
PUTATR	E6E4	E6C8	○	フィールド中のバッファポインタ以降の領域のアトリビュートを設定する
PUTOSB	E700	E6E4	○	PUTのSBA(Set Buffer Address)オーダルーチン
PUTORC	E731	E715	○	PUTのRC(Repeat Character)オーダルーチン
PUTDAT	E745	E729	○	共有RAM上の出力文字列中から1文字読み込む
PUTOBS	E76B	E74F	○	PUTのBS(Back Space)オーダルーチン
BTOC	E77E	E762	○	バッファポインタのワークエリアの内容をカーソルのワークエリアにコピーする
CTOB	E787	E76B	○	カーソルのワークエリアの内容をバッファポインタのワークエリアにコピーする
PUTOTB	E793	E777	○	PUTのHT(Horizontal Tab)オーダルーチン
TABCHK	E79D	E781	○	TABの停止位置かどうかのチェック
PUTINC	E7BA	E79E	×	バッファポインタを1つ更新する
PUTDW	E7DE	E7AF	×	バッファポインタのY座標を1つ増やす
ESCSEQ	E7FB	―――	×	PUTのエスケープシーケンスオーダルーチン
SCROLL	E82E	E7DA	×	画面を1行スクロールアップする
COPLOP	E8F1	E89D	○	ブロック転送(8バイトずつ)ルーチン
PUTORD	E908	E8B4	−	PUTのオーダサブルーチンへのジャンプテーブル
GET000	E930	E8DC	×	GETメインルーチン
GETABT	―――	E927	×	GETメインルーチンのアボート(Abort)ルーチン
GETRNG	E975	E92F	○	ウィンドウ画面のコンソールバッファ上での範囲を求める
GETDT	E98A	E944	×	入力された文字を表示し，コンソールバッファへ格納する
GETDT1	E99D	E957	○	入力された文字の表示・格納を行う（GETINS用)
GETINS	E9D2	E98C	×	挿入処理を行い，入力された文字の表示・格納を行う
CSRL	EA14	E9C3	○	カーソルを1つ右に移動する
CSLL	EA1F	E9CE	○	カーソルを1つ左に移動する
CSUL	EA29	E9D8	○	カーソルを1つ上に移動する
CSDL	EA37	E9E6	○	カーソルを1つ下に移動する
CSR	EA44	E9F3	○	カーソルを1つ進める
CSL	EA60	EA0F	○	カーソルを1つ前に戻す

※○でアルゴリズムの相違無し，Xで相違有り

ラベル名	FM－7	FM－8	※相違	機　　能
GETENC	EA79	EA28	○	キャラクタバッファアドレスから，カーソルのXY座標を求める
GETENA	EA7F	EA2E	○	アトリビュートバッファアドレスから，カーソルのXY座標を求める
GETEN	EA83	EA32	○	ウィンドウ画面の相対位置から，カーソルのXY座標を求める
CSHOME	EA94	EA43	○	カーソルを画面の先頭に移動する
CSHOM0	EA97	EA46	○	次のフィールドの先頭へカーソルを移動する
GETFFD	EAB0	EA5F	○	カーソルを後方のフィールドの先頭へ移動する
GETBFD	EAB8	EA67	○	カーソルを前方のフィールドの先頭へ移動する
GETBHM	EABE	EA6D	○	前のフィールドの先頭へカーソルを移動する
GETCLR	EAD2	EA81	○	ウィンドウ画面を消去し，入力処理を終了させる
INSERT	EAE0	EA8F	○	インサートモードの切換えを行う
ELKEY	EAEC	EA9B	○	次のフィールドの手前までの文字を消去する
GETDUP	EB21	EAD0	○	DUP動作を行う
TAB	EB2B	EADA	○	TAB動作を行う
TABDUP	EB2C	EADB	○	TABの停止位置まで，空白または上の行の内容を出力する
CSRW	EB56	EB05	○	カーソルを後方の英数文字列の先頭へ移動する
CSRWS	EB75	EB24	○	英数文字を後方に検索していく
CSWCK	EB8C	EB3B	○	英数文字かどうかチェックする
CSLW	EBA4	EB53	○	カーソルを前方の英数文字列の先頭へ移動する
CSLWS	EBB9	EB68	○	英数文字を前方に検索していく
CSWCKT	EBD2	EB81	○	英数文字検索用のデータ
GETBS	EBDC	EB8B	×	BS(Back Space)処理を行う
GETDEL	EBE8	EB97	×	DELキーに対する処理を行う
GETCAN	EC3E	EBE7	○	入力処理をキャンセルする
GETCR	EC41	EBEA	○	入力処理を終了させ，最上段の変更フィールドを転送する
GETC	EC46	EBEF	×	GETC(未転送)コマンドルーチン
GETJMP	ECAC	EC58	－	GETのオーダサブルーチンへのジャンプテーブル
GETBAD	ECEC	EC98	○	Get Buffer Addressコマンドルーチン
CGBLK2	ECF4	ECA0	○	Get Character Blockコマンドルーチン
CGBLK1	ECF5	ECA1	○	Get Character Block 1コマンドルーチン
CGBDT	ED0F	ECBB	○	Get Character Block用1バイト出力（共有RAMへ）

※○でアルゴリズムの相違無し，×で相違有り

ラベル名	FM－7	FM－8	※相違	機　　　能
CBINIT	ED23	ECCF	○	Get/Put Character Block用初期設定ルーチン
CBLADR	ED54	ED00	○	バッファアドレス生成，座標の更新，終了判定
CBLEND	ED6C	ED18	○	Get/Put Character Blockエンディングルーチン
CPBLK2	ED71	ED1D	○	Put Character Block 2 コマンドルーチン
CPBLK1	ED72	ED1E	○	Put Character Block 1 コマンドルーチン
CPBDT	EDA5	ED51	○	Put Character Block用1バイト入力（共有RAMから）
TABSET	EDBC	ED68	○	TAB　SETコマンドルーチン
CNSCTL	EDC8	ED74	○	Console Controlコマンドルーチン
CNSINT	EDCB	ED77	○	コンソール制御ルーチン(リスタート用)
CURON	EDD3	ED7F	○	カーソルの表示を開始させる
CUROFF	EDE3	ED8F	○	カーソルの表示を禁止する
CURSET	EDE8	ED94	×	カーソルの位置を設定する
CURSCL	EDED	ED9D	○	カーソルの表示を中止し，カーソルをクリアする
CURCLR	EDEF	ED9F	○	カーソルをクリアする
CURSTP	EE01	EDB1	○	カーソルの表示を中止する
CURGO	EE07	EDB7	○	中止してあったカーソルの表示を再び開始させる
INDICT	EE0E	EDBE	○	PFキー表示ルーチン
INDI	EE33	EDE3	○	PFキー1行表示ルーチン
INDSPC	EE43	EDF3	○	空白を表示する(PFDSP用)
INDCHR	EE45	EDF5	○	文字を表示する(PFDSP用)
PFDSP	EE54	EE04	○	1つのPFキーを表示する
CHRLN	EE72	EE22	○	Character Lineコマンドルーチン
LINE	EE98	EE48	○	LINEコマンドルーチン
LINE10	EEC7	EE77	○	四角形の枠を描く
LINE20	EEF7	EEA7	○	四角形を塗り潰す
LINES	EF1A	EECA	○	ラインの実行ルーチン
LNPOIN	EFA0	EF50	○	ラインのポイントおよびキャラクタライン1文字出力
LINEY	EFCA	EF7A	○	Y軸に平行なラインを描く
LINEX	EFE6	EF96	○	X軸に平行なラインを描く
LNRSID	F05A	F00A	○	X軸に平行なラインの右サイドの出力

※○でアルゴリズムの相違無し，×で相違あり

ラベル名	FM－7	FM－8	※相違	機　　　能
LNLSID	F05D	F00D	○	X軸に平行なラインの左サイドの出力
LNXFNC	F07E	F02E	○	ラインを行うサブルーチンへのフック用のジャンプベクトルデータ
LNAND～	F092	F042	○	ライン出力ルーチン群
CHAIN	F0C4	F074	○	CHAINコマンドルーチン
SYMBOL	F103	F0B3	○	SYMBOLコマンドルーチン
SYMB10	F16D	F11D	○	SYMBOLメインループ(SYLOOP)
CHANGE	F21B	F1CB	○	Change Colorコマンドルーチン
GBLK1	F286	F236	×	Get Block 1 コマンドルーチン
GBINT	F2CB	F2C0	○	Get Block 1/2の初期設定ルーチン
CLEXT	F2ED	―――	×	カラーエクステンドテーブルの作成（Get Block 1用）
GBSCL	F301	F2E2	○	CLEXTサブルーチン
GBPUT	F308	F2E9	○	Get Block用1バイト出力(共有RAMへ)
GBPLA	F325	F306	○	Get Block行データの変形・出力
GBLK2	F3B6	F397	○	Get Block 2 コマンドルーチン
GBLK2S	F3CF	F3B0	○	Get Block 2 サブルーチン
GBOINT	F3FC	F3DD	○	Get Blockの出力初期設定ルーチン
POINT	F412	F3F3	○	POINTコマンドルーチン
PBLK1	F446	F427	○	Put Block 1 コマンドルーチン
PB1050	F45D	F43E	○	Put Block 1 メインループ
PB1FNC	F48E	F46F	－	Put Block 1 表示ルーチンへのフック用ジャンプベクトルデータ
PBINT	F4A6	F487	○	Put Block 1/2の初期設定ルーチン
PBSHFT	F50D	F4EE	○	Put Blockの行データの入力・変形
PBGET	F56D	F54E	○	Put Block用1バイト入力(共有RAMから)
PBLK2	F589	F56A	○	Put Block 2 コマンドルーチン
PB2SUB	F5B1	F592	○	Put Block 2 サブルーチン
PB2LRM	F5D4	F5B5	○	行データの両サイドのバイトのマスク処理
PB2SET～	F5E1	F5C2	○	Put Block 2 用表示ルーチン群
PB1OR～	F62B	F60C	○	Put Block 1 用表示ルーチン群
LNXOR～	F631	F612	○	ライン表示ルーチン(RAMを参照するもの)群
PB2FNC	F66E	F64F	－	Put Block 2 表示ルーチンへのフック用ジャンプベクトルデータ

※○でアルゴリズムの相違無し，×で相違有り

ラベル名	ＦＭ－７	ＦＭ－８	※相違	機　　　能
ＧＣＵＲＳ	Ｆ６７Ｄ	Ｆ６５Ｅ	×	Graphic Cursorコマンドルーチン
ＧＣ０００２	Ｆ６Ｆ６	Ｆ６ＤＡ	×	Graphic Cursorメインループ(ＧＣＬＯＯＰ)
ＧＣ０００４	Ｆ７１８	Ｆ７０６	○	リターンキー処理(ＧＣＲＴＫＹ)
ＧＣＩＲＱ	Ｆ７４０	Ｆ７２Ｅ	○	キャンセルルーチン
ＧＣＲＴＳ	Ｆ７４６	Ｆ７３４	○	エンディングルーチン
ＧＣ０００Ｂ	Ｆ７５６	Ｆ７４４	×	グラフィックカーソル移動キーの処理（ＧＣＭＯＶＥ）
ＧＣＴＡＢＬ	Ｆ７９Ａ	Ｆ７６９	×	Graphic Cursor用ジャンプテーブル
ＧＣＵＰ２０	Ｆ７Ａ８	Ｆ７Ａ９	○	グラフィックカーソルを２０ドット上方へ移動
ＧＣＤＷ２０	Ｆ７Ｂ５	Ｆ７Ｂ６	○	グラフィックカーソルを２０ドット下方へ移動
ＧＣＨＯＭＥ	Ｆ７ＣＦ	Ｆ７Ｄ０	○	グラフィックカーソルを初期位置へ移動
ＧＣＵＰ	Ｆ７Ｅ０	Ｆ７Ｅ１	○	グラフィックカーソルを１ステップ上方へ移動
ＧＣＤＷ	Ｆ７ＥＣ	Ｆ７ＥＤ	○	グラフィックカーソルを１ステップ下方へ移動
ＧＣＲ２０	Ｆ７ＦＣ	Ｆ７ＦＤ	○	グラフィックカーソルを２０ドット右へ移動
ＧＣＳＦＴＲ	Ｆ８０５	Ｆ８０６	○	グラフィックカーソルを１ステップ右へ移動
ＧＣＳＦＴ２	Ｆ８１Ａ	Ｆ８１Ｂ	○	グラフィックカーソルパターンレジスタ右ｎドット移動
ＧＣＬ２０	Ｆ８３Ｆ	Ｆ８４０	○	グラフィックカーソルを２０ドット左へ移動
ＧＣＳＦＴＬ	Ｆ８４Ａ	Ｆ８４Ｂ	○	グラフィックカーソルを１ステップ左へ移動
ＧＣＢＤＲＹ	Ｆ８８０	Ｆ８８１	○	境界条件の判定
ＧＣＳＡＶＥ	Ｆ８ＥＣ	Ｆ８ＥＤ	○	画面情報の退避
ＧＣＳＡＶ１	Ｆ９１０	Ｆ９１１	○	画面情報の退避サブルーチン
ＧＣＤＩＳＰ	Ｆ９４Ｄ	Ｆ９４Ｅ	○	グラフィックカーソルの表示
ＧＣＤＯＲ	Ｆ９８６	Ｆ９８７	○	グラフィックカーソルのＯＲ処理
ＧＣＤＡＮＤ	Ｆ９Ｄ２	Ｆ９Ｄ３	○	グラフィックカーソルのＡＮＤ処理
ＧＣＲＳＡＶ	ＦＡ２８	ＦＡ２９	○	画面情報の復帰
ＧＣＲＡ１	ＦＡ４Ｃ	ＦＡ４Ｄ	○	画面情報の復帰サブルーチン
ＰＡＩＮＴ	ＦＡ８Ａ	ＦＡ８Ｂ	×	ＰＡＩＮＴコマンドルーチン
ＰＡＰＡＩＮ	ＦＡＥ０	ＦＡＦ６	○	ＰＡＩＮＴ初期化ルーチン
ＰＡＳＥＡＲ	ＦＢ２２	ＦＢ３８	○	ＰＡＩＮＴ右塗り潰し
ＰＡＬＩＮＬ	ＦＢ６Ｃ	ＦＢ８２	○	ＰＡＩＮＴライン実行ルーチン
ＰＡＳＥＡＬ	ＦＢ８６	ＦＢ９Ｃ	○	ＰＡＩＮＴ左塗り潰し

※○でアルゴリズムの相違無し，×で相違有り

ラベル名	FM－7	FM－8	※相違	機　　　　能
PASTKO	FBEC	FC02	○	PAINT情報をスタックから取り出す
PASTKI	FC30	FC46	○	PAINT情報をスタックに入れる
PASEA	FC57	FC6D	○	PAINT上下の移動
PABGET	FC80	FC96	○	境界色を調べる
GBLKDT	FC86	――――	○	指定色を調べる(Get Block 1 より)
PASTP1	FCC2	FCDC	○	右の境界色サーチする
PASTP2	FCC7	FCE1	○	左の境界色サーチする
PASTP3	FCCC	FCE6	○	右の境界色でないところをサーチする
PASTP4	FCDD	FCF7	○	左の境界色でないところをサーチする
INKEY	FD5C	FD76	×	INKEYコマンドルーチン
KEYGET	FD6E	FD8E	×	キー入力待ちを行いキーコードを読取る
KEYIN	FD70	FD90	×	キーコードを読取る
KEYIN0	FD76	FD90	×	キーコードを読取る
INCKBX	FD95	――――	×	キー入力バッファの読取り用ポインタを補正する
KBFERA	FD9E	――――	×	キー入力バッファをクリアする
KEYIRQ	FDAC	FDB9	×	キー割込みの処理を行う
KSTBUF	FDE6	――――	×	入力されたキーコードをキー入力バッファへ格納する
KEYR2	――――	FDDE	×	PFキーを除くキーのキーコードを読み取る
FUNKEY	FE02	FDE5	○	Define String of PFコマンドルーチン
CLRARA	FE34	FE17	○	特定の領域をクリアする
FUNADR	FE3D	FE20	○	PFキー定義テーブルのアドレスを求める
GSTRPF	FE46	FE29	○	Get String of PFコマンドルーチン
PFINT	FE61	FE44	○	Interrupt Controlコマンドルーチン
SETTIM	FE7C	FE5F	○	Set Timerコマンドルーチン
REDTIM	FEA4	FE87	○	Read Timerコマンドルーチン
TMCT	FEB2	FE95	○	Set/Read Timerエンディングルーチン
TIMNMI	FEBF	FEA2	○	NMI(タイマ割込み)サービスルーチン
TIMER	FFC8	FEAB	○	各タイマレジスタの更新，カーソルの点滅，アテンション割込みの発生
TMDAT	FF6B	FF4E	○	タイマ時刻レジスタの各バイトの桁あふれ処理のための数値
TEST	FF6F	FF52	×	TESTコマンドルーチン

※○でアルゴリズムの相違無し，×で相違有り

5－2　ＦＭ－７／ＮＥＷ７／７７　内部ルーチンの比較

ＦＭ－７と７７，ＮＥＷ７では，ほとんどのルーチンで違いはありません（ＤＩＳＫ，ＢＩＯＳを除く）．しかも直したところは，ほとんどがバグと思われるところで，アドレスは変化しないように直してあります．ですからＦＭ－７の内部ルーチンを使っているプログラムをＦＭ－７７，ＮＥＷ７で動作させても暴走するということはまずないでしょう．

では，主な違いについて述べます．

★ＦＭ－ＮＥＷ７のＦＭ－７との比較 （ＦＭ－ＮＥＷ７について）

○ＭＯＮでフィールドの設定を行っている（＄ＡＣ０６～）．

○ＬＰＯＳで数値型であれば文字列の実行番地を呼び出す（＄Ｃ９４７～）．

○ＲＥＮＵＭのバグが直っている（＄ＡＢ７Ｃ～）．

○ＣＨＡＩＮ実行中にBreakキーが押されてもうまく処理できるようにしてある（＄Ｄ６８Ｄ～）．

○プリンタ１文字出力時にＣＲ，ＬＦを出力する場合にポインタを０にするようにしてある（＄Ｄ６１９）．

○タイマをセットしてからＰＦキーインタラプトを初期化するようにしてある（＄Ｄ４Ａ７～）．

○ＰＬＡＹ文に改良を加えてある．

などが挙げられます．

★ＦＭ－７７のＦＭ－ＮＥＷ７との比較（ＦＭ－７７について）

○ＢＩＯＳを変えている（これらは大幅に行っている）．

○ＲＥＮＵＭの改良をしている．

○ＶＡＬ関数で，ＶＡＬ（"＆ｈ"＋"ＥＥＥＥ"）などのようにｈの小文字が使える．

○文字列→数値，数値→文字列変換のバグを直している．

などが挙げられます．

共　著　　箕原辰夫　　菊地　寿　　南　泰浩

発行者　牧谷秀昭
発行所　秀和システムトレーディング株式会社
郵便番号１０７　東京都港区南青山２－１９－５　関原ビル

SHUWA SYSTEM TRADING CO., LTD.
SEKIHARA BLDG,
2-19-5 MINAMIAOYAMA, MINATO-KU, TOKYO, 107 JAPAN

印刷所　東京スガキ印刷株式会社

発行日　１９８４年８月１日

秀和システム

秀和システムトレーディング株式会社
ISBN4-87966-023-X C0000 ¥2800E